PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04Q 7/20 — A1

(11) International Publication Number: WO 96/09731

(43) International Publication Date: 28 March 1996 (28.03.96)

(21) International Application Number: PCT/US95/12318

(22) International Filing Date: 21 September 1995 (21.09.95)

(30) Priority Data:

310,908	22 September 1994 (22.09.94)	US
354,200	12 December 1994 (12.12.94)	US

(71) Applicant (*for all designated States except US*): ACCESSLINE TECHNOLOGIES, INC. [US/US]; 11201 S.E. 8th Street, Bellevue, WA 98004 (US).

(72) Inventors; and

(75) Inventors/Applicants (*for US only*): FULLER, Robert, M. [US/US]; 2624 260th Place, S.E., Issaquah, WA 98027 (US). BERG, Richard, P. [US/US]; 2071 Hopewell Court, Thousand Oaks, CA 91360 (US). KRANZLER, Daniel, R. [US/US]; 14040 N.E. 6th Place, Bellevue, WA 98007 (US).

(74) Agents: PACIULAN, Richard, J. et al.; Ladas & Parry, Suite 2100, 5670 Wilshire Boulevard, Los Angeles, CA 90036 (US).

(81) Designated States: AM, AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LT, LU, LV, MD, MG, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TT, UA, UG, US, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG), ARIPO patent (KE, MW, SD, SZ, UG).

Published

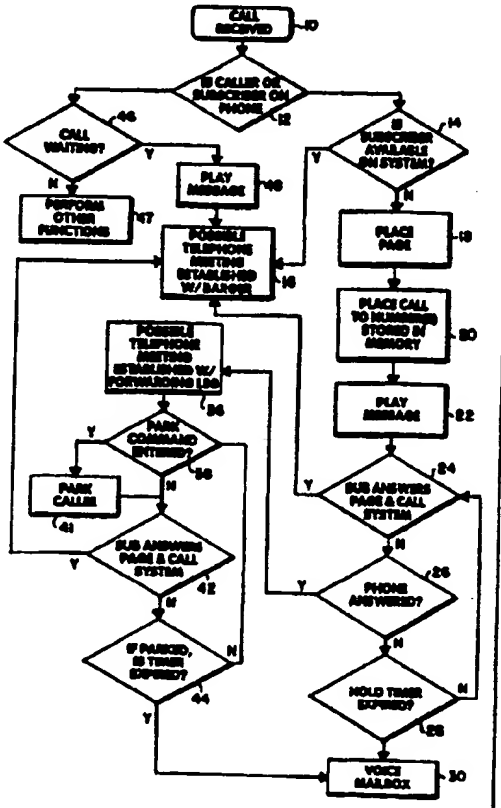
With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: COMPUTER CONTROLLED PAGING AND TELEPHONE COMMUNICATION SYSTEM AND METHOD

(57) Abstract

A method and an apparatus for placing a caller into telecommunication with a subscriber. A call from the caller, directed to the subscriber, is detected (10) and, in response thereto, a page is initiated to the subscriber and at the same time a forward leg call is initiated to a stored telephone address. If the call on the forward leg is answered, the party answering that call is placed in telecommuncation with the caller. A call from the subscriber is also detected and the subscriber is placed in telecommunication with the caller together with any party on a forward leg call.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria
AU	Australia
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CS	Czechoslovakia
CZ	Czech Republic
DE	Germany
DK	Denmark
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgystan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakhstan
LI	Liechtenstein
LK	Sri Lanka
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SI	Slovenia
SK	Slovakia
SN	Senegal
TD	Chad
TG	Togo
TJ	Tajikistan
TT	Trinidad and Tobago
UA	Ukraine
US	United States of America
UZ	Uzbekistan
VN	Viet Nam

COMPUTER CONTROLLED PAGING AND TELEPHONE COMMUNICATION SYSTEM AND METHOD

Cross Reference to Related Applications

This application is related to the Intelligent Telephone Control System disclosed in U.S. Patent Application 08/354,200 of Robert M. Fuller et al as inventors and which is assigned to the applicant herein (the assignee in the United States). The aforementioned U.S. Patent Application No. 08/354,200 is hereby incorporated herein by reference.

Technical Field

The present invention relates to a computer controlled signaling and telephone communication system and method for notifying a subscriber of a telephone call placed by a caller to the subscriber.

Background of the Invention

Telephone systems are well known and, indeed, a number of techniques are known in the prior art to facilitate connecting a caller (i.e. the party initially placing the telephone call) with a telephone subscriber (i.e. the party whom the caller is trying to contact by telephone). Of course, a very well known way to accomplish this end is to have a telephone ring at a subscriber's presumed location.

It is well known, however, that subscribers move about during the day, and therefore, other techniques

have been employed to try to place callers into telephone communication with subscribers. These other techniques include paging systems and voice mail systems. However, both paging systems and voice mail systems typically suffer a drawback in that the subscriber must receive or pick up a message either from the paging system or from a voice mail system, or some combination of the two, and thereafter, place a telephone call to the caller in order to be placed into direct telephone communication with the caller.

A problem with this approach is that it is not particularly efficient. By the time the subscriber receives the page and responds to it, the caller may have left the place from which they were telephoning or may be at an unknown location, in which case, the subscriber is frustrated in attempting to make telephone contact with the caller.

One solution to this problem, as suggested in the prior art, is to provide a "meet-me" service in which a caller is placed on hold at the same time the subscriber is paged to a telephone. The subscriber then telephones a telephone system and the system, upon recognizing the call from the subscriber, connects the subscriber with the original caller who has been placed on hold. This "meet-me" system is described in U.S. Patent Nos. 4,642,425 (to Guinn) and 5,151,929 (to Wolf). When the subscriber is connected to the caller this action is sometimes known as "barging in".

This prior art "meet-me" system, while improving telephone communications, still suffers certain drawbacks. For example, in a modern office, people do

not necessarily stay put at their desk. They need to interface with other people in their office, and therefore, discussions may be held in offices other than their own or in conference rooms and; of course, they may leave the office to go to lunch, the bathroom or run errands. Some subscribers have a tendency to use the prior art "meet-me" system a great deal of time. While it is possible with current telephone systems to switch the "meet-me" service on and off, it does require that a subscriber input certain codes at a telephone to switch off the "meet-me" service and then key in certain codes at a telephone to turn the "meet-me" service back on. If the subscriber uses the "meet-me" system in the manner described, they are supposed to telephone the system to turn off the meet me mode whenever they arrive at the office and turn it back on whenever they leave the office. That might work well in theory, but it does not work well in practice because people forget to switch their telephone service, either when leaving the office or returning to the office; or alternatively, they will leave their telephones in the "meet-me" mode 100% of the time.

A problem which arises when a subscriber uses the "meet-me" mode is that it takes a relatively long time for a caller to make telephone contact with the subscriber, since even if they are sitting at their desk, the "meet-me" system first pages them, requiring them to pick up their telephone and input certain numbers in order to gain access to a telephone switch where they eventually meet their caller. A telephone call which might otherwise take only 15 or 20 seconds to complete, can take three or four times as long when the subscriber leaves their telephone in the "meet-me" mode.

It is, therefore, one object of the present invention to provide and improve this "meet-me" service. With the improved "meet-me" service described herein, a telephone subscriber may keep his or her telephone in a "meet-me" mode most of the time, but the system, in addition to paging the subscriber, also places a telephone call from the caller to one or more expected locations. Moreover, even if the paging feature is not utilized, it can be very advantageous to be able to try to contact the subscriber at a plurality of locations at more or less the same time automatically. For example, a telephone call placed to a subscriber can be redirected as a plurality of calls placed more or less simultaneously to the subscriber's home, car telephone and office (for example). The caller can be connected to whoever answers these calls, while the other unanswered calls continue to ring. If the paging feature is utilized, then the subscriber can respond to a page by calling into the system and be connected to the caller regardless of whether any of the plurality of calls were answered.

The present invention provides a system and a method whereby an incoming telephone call is handled by a telephone system controller. The controller is preferably a programmed device, much like a computer (in fact in the preferred embodiment it is a computer which has been adapted in certain ways to enable it to communicate with the Public Switched Telephone Network (PSTN)). The controller initiates a plurality of contemporaneous communications seeking the subscriber. This plurality of contemporaneous communications may include a plurality of independent telephone calls to

different destinations or at least one such telephone call and a contemporaneous page.

Another problem which prior art telephone systems have relates to allowing the subscriber to monitor a telephone call thereby giving the subscriber the opportunity to answer an incoming call or to allow the caller to be handled by an answering machine. Indeed, modern answering machines allow the user to listen to (i.e. monitor) an incoming call in order to decide whether to answer the call immediately or to allow the answering machine to complete its task. As is well known, however, telephone subscribers are apt to receive more than one telephone call at a time, but conventional answering machines can only deal with a single call at a time (unless the cost of a second telephone line and a multi-line answering machine are borne). Several technologies have been developed to provide smoother telephone services which enable a subscriber to deal with an incoming telephone call if they happen to be on the line with another party at the time a telephone call is received. The "Call Waiting" service offered by many telephone companies allows a caller to receive a second telephone call after having answered a first telephone call. Telephone companies have also offered voice mail services to their subscribers. However, those services do not allow a subscriber to monitor an incoming call like a telephone answering machine does, but those service do have the advantage of being able to take a message from more than one caller at the same time or to take a message while the subscriber is engaged with another party on the telephone. As a result, these technologies (conventional answering machines and telephone company provided voice mail service) have

competed with each other since they each provide service needs which the other cannot provide.

In addition to or complimentary to the improved "meet-me" system mentioned above, the present invention provides a voice mail system to which an incoming call may be directed. The incoming caller may be routed to the voice mail system directly or after having been processed by the system in a manner which will be described in greater detail below. The subscriber can then connect to the voice mail system in a "monitoring mode" which allows the subscriber to overhear the message being left by the calling party for the subscriber. If they wish, the subscriber may barge into the telephone call so that they may speak with the calling party, after having preferably given the calling party notice that the subscriber is barging in. The subscriber may barge in from a telephone instrument which is called by the system or after the subscriber has telephoned into the system, for example, in response to a page.

These and other features of the invention will become even more clear upon reading the following detailed description in conjunction with the attached drawings.

Brief Description of the Drawings

Figure 1 is a diagram of a system of the type which may be used to embody the present invention;

Figure 2 is a flow diagram showing the processing capabilities of the call processing facility of the

system of Figure 1;

Figure 3 shows how the system interacts with callers, subscribers and other persons;

Figures 4a - 4k provide a detailed flow diagram of a preferred embodiment of the invention - the flow diagram includes a Main Task in Figure 4a, an Inbound Task in Figures 4b and 4b(1) - 4b(7), a Paging Task in Figure 4c, a Barger Disconnect Task in Figure 4d, a Forward Leg Task in Figures 4e(1) - 4e(4), a Forward Leg Disconnect Task in Figure 4f, a Voice Mail Task in Figure 4g, a Voice Mail Disconnect Task in Figure 4h, a Caller Disconnect Task in Figure 4i, a Timer Task in Figure 4j and a Record Task in Figure 4k;

Figure 5 is a schematic diagram of a hardware environment for the system which connects with one type of telephone company circuits;

Figure 6 is a schematic diagram of a hardware environment for the system which connects with telephone company T-1 type circuits.

Brief Description of the Appendices

Appendix I is a source code listing of the Main Program written in VOS (Voice Operating System).

Appendix II is a source code listing of an Inbound Task used by the Main Program.

Appendix III is a source code listing of a Forward Task used by the Main Program.

Appendix IV is a source code listing of a Page Task used by the Main Program.

Appendix V is a source code listing of a Voice Mail Task used by the Main Program.

Appendix VI is a source code listing used by Inbound Task for phone number playback.

Appendix VII is a subroutine listing defining constants and variables used by all Tasks.

Appendix VIII is a subroutine listing defining constants used by the Dianatel hardware switch of the preferred embodiment of the present invention.

Appendix IX is a subroutine listing used for conferencing by all Tasks other than the Main Program.

Appendix X is a listing of ASCI text files which define various VOS user setup parameters.

Detailed Description

Overview

Figure 1 illustrates, in block diagram form, a Telephone Control System that may be used to enhance the accessibility of it's subscribers to callers. As is shown, the Telephone Control System 40 connects with the PSTN 50 via facilities 51. The Telephone Control System 40 may control a switch 54, causing it to connect

incoming and outgoing telephone circuit, e.g., trunks or lines 51.

In an alternate embodiment, the switch 54 is actually part of the PSTN 50. In this embodiment, the facilities 51 must be capable of transmitting switch control signals from the Telephone Control System 50 to the switch 54. An example of this type of facility is a CENTREX line, which allows the transmission of switch control signals in the form of `hookswitch flashes` and touch tones to initiate call-conferencing, call-transfer, barge-in, speed-dialing, plus all other CENTREX functions controlled by this method. A variation of the CENTREX facility is a CENTREX two-way DID trunk, which not only has the `hookflash` capability, but also provides the called number in the form of Direct-Inward-Dialing digits. This is one form of facility 51. Another variation of the CENTREX facility provides the called number via a separate data-link known as Simplified Message Desk Interface (SMDI). Other facility type are ISDN primary rate or ISDN basic rate interfaces which provide all of these capabilities plus many other in addition to two voice compatible channels and a packet channel.

In the preferred embodiment, the switch 54 is part of the Telephone Control System 40. In this embodiment, the facilities 51 need only include standard DID trunks for the incoming calls, and standard outgoing trunks. The control system 40 controls switch 54 directly, causing it to connect paths between various incoming and outgoing trunks as required.

Again referring to Figure 1, the Telephone Control System 40 may also connect to the PSTN 50 via standard

phone lines 52, for purposes of communicating with the Paging System 60. The Paging System may be any of the commonly known paging systems such as those comprised of transmitters such as Motorola's PACE or Quintron model QT250B and paging terminals such as Glenayre model GL3000XL or BBL System 3, which send encoded messages via radio frequency to cause a unique pager, or beeper, worn by a paging system subscriber, to sound an alert, produce a message in a display, activate a light, vibrate, or produce any of a variety of other alerting mechanisms. Typically, these paging systems will cause a pager to be alerted in response to another individual dialing a phone number which corresponds to that individual's pager. This phone number may be routed via the PSTN 50 to a paging system 60 (unless the paging terminal is directly coupled to system 40), which paging system 60 in turn determines, typically via DID digits, who the call is intended for, and then sends a radio frequency message to alert that individual's pager. To cause a subscriber's pager to be activated, the Telephone Control System 40 then dials the phone number that corresponds to the subscriber's pager. Although not described in this preferred embodiment, it is anticipated that the Telephone Control System 40 could also interface to a paging system directly via a dedicated data link.

An additional facility 53 preferably connects the Telephone Control System 40 to the PSTN 50. This facility is a trunk which provides the Automatic Number Identification (ANI) of the calling party or the CLID (Calling Line IDentification) of the calling party. An example of such a trunk is the Feature Group D (FGD) trunk which is commonly used by interexchange carriers.

The interexchange carriers use the ANI information to properly bill the calling party. The Telephone Control System 40 may use this ANI and/or CLID information to differentiate between calling parties and subscribers, and for other purposes described herein. Although not described in the preferred embodiment, it is anticipated that other types of facilities which provide ANI information may also be used for this purpose. An example of another type of facility which provides ANI is a CENTREX line with an SMDI data link, which is now available from several types of central offices. The SMDI data link is capable of passing both the called party number and the calling party number (ANI/CLID). In addition, ISDN primary rate and basic rate interfaces supply such information.

To aid in the discussion of the illustrative examples which follow, Figure 1 also shows a subscriber's home 15, with a home phone 16; a subscriber's office 17, with an office phone 18; a cellular telephone system 19, which interfaces to a subscriber's car-phone 20; a factory 21, with a factory phone 22; a pay telephone 23; a subscriber 24 with pager 25; and a caller's telephone 26. A voice mail system 90 is depicted as a stand-alone system. Those skilled in the art will appreciate that the voice mail system 90 may be embodied in telephone control system 40 instead, or elsewhere in the overall system.

The illustrative examples which follow are intended only to clarify some of the concepts, features, and objects of the invention, and do not define the scope of the invention. In this embodiment, the apparatus and method of the invention control switch 41 which may be

located, for example, in the central office of a telephone company. Alternatively, and preferably, a controlled switch 41 is included with the apparatus of the present invention, which, in turn, may be located either on the premises of the telephone company switch or elsewhere, including the premises of a business which make use of the system. In another embodiment the invention is embodied in a telephone control system 40 which may be placed upstream of a private branch exchange (PBX) having direct inward dialing (DID) capabilities.

Before describing the preferred embodiment in detail, a general overview of how the invention may be practiced is first described with reference to the flow diagram of Figure 2. This flow diagram is straight forward and is intended to give an overview of how the system handles telephone calls. A more robust flow diagram is set forth in Figures 4a - 4k which shows how the preferred embodiment of the system would function in a multitasking environment.

Referring to Figure 2, a call is received at block 10 and a decision is made at step 12 as to whether or not the person placing the call is a subscriber of the system or is a caller who is trying to reach a subscriber 65 (Figure 1). The location of the subscriber may be unknown. The subscriber may be at home 79, at the office 77, in their automobile or elsewhere. The telephone control system will attempt to use telephone system resources to place the subscriber in telephone contact with a caller located at a phone 6.

Callers and subscribers can be differentiated in a

number of ways. Thus the decision at step 12 can be made by various means, including (i) examining the Calling Line I.D. (CLID) of the party placing the call to determine if the caller is a known subscriber and/or (ii) examining the Direct Inward Dial (DID) digits received from the Public Switched Telephone Network (PSTN) 50 (this embodiment assumes that the subscriber has one telephone number which he or she uses to contact the system and a different telephone number which callers use to contact the subscriber) and/or (iii) whether a special code (i.e. a personal identification code or PIN) is input by the person placing and/or receiving the call to identify themselves as a subscriber. Since the CLID is not always available and since forcing the subscriber to use a PIN makes the marketing of a telephone service more difficult, the currently preferred manner of dealing with this problem is technique (ii) mentioned above, namely, one telephone number is dedicated to each subscriber for the subscriber to use when calling the system to pick up messages or, in the context of the present invention, to meet a calling party who has called the system trying to contact them, and a second telephone number is dedicated to each subscriber for callers to use when calling the subscriber via the system. In any event, it is contemplated that incoming calls can be differentiated between subscribers of the system and callers (who are trying to contact subscribers).

Assuming that a caller is on the line trying to reach a subscriber, a test is preferably made at step 14 to determine whether or not the subscriber is already available on the system. Whether or not this particular test is made at this stage, depends in part, on whether

or not the particular hardware which is utilized will support the making of such a test and whether the additional complexity of including this test is deemed to be worthwhile. It is believed that this test should preferably be performed. If the subscriber is already available on the system at the time the caller calls, the processing branches to block 16 where a telephone connection is established between the caller and the subscriber. The subscriber may already be on the system if, for example, the system includes a voice mail system and therefore the subscriber may already be on the system at the time the caller calls for the purpose of picking up his or her voice mail messages. Alternatively the subscriber may be engaged in a conversation with another caller. In that case, when the new caller reaches the system, call waiting treatment is preferably applied to the subscriber's circuit.

At block 16, when a telephone meeting is established, that means that a connection is made between the subscriber who has dialed the system on one line with a caller who has also dialed the system on a different telephone line. However, the connection can be either a full duplex connection (in which case both parties can speak with each other) or it can only be a half duplex connection (in which case the subscriber can listen to what the caller says, but cannot speak with the caller). This half duplex mode is used if the subscriber wishes to monitor the call before making a decision to speak with the calling party. Typically, the calling party might be conveyed to a voice mail system where the subscriber can "eavesdrop" on the message which the calling party is leaving for the subscriber. Then, if desired, the subscriber can signal the system

to switch to full duplex communication after the system has preferably first advised the calling party that the subscriber is entering the conversation. The voice mail system 90 is preferably set up to pause when the subscriber barges in, so that recording then stops. However, the voice mail system 90 may also be designed to allow it to monitor for a code which would cause is to resume recording. This would be convenient for the parties if they had some particularly good ideas which they would like to record and conventional note-taking pads are either unavailable or inconvenient to use (such as when driving an automobile).

If the subscriber is not available at step 14, or if that test is not made, the call processing falls through to block 18 where a page is placed to the subscriber. The page may be via a conventional radio communication paging system 60 to a device 61 which signals the subscriber by making beeping sounds, or vibrating, or otherwise signaling the subscriber that they are being paged. Alternatively, the page may be by other means, including a computer generated audible voice page delivered in a suite of offices or rooms 70 by means of speakers 71 advising a party that they have a call. Such a page can be delivered directly to speakers built into many telephones and a subscriber, after being so paged, can go directly to the nearest telephone, call the system (if needed), enter a PIN code and be connected to the calling party. Some means of signaling the subscriber that they have a call is provided, so that they know to access the system and make a connection with the caller.

More or less simultaneously, the system also places

a call to a telephone where the subscriber 65 is expected to be located, for example, to phone 76 at office 77 or a phone 72 in a room 70 in a building. This call may be placed via a PBX to a telephone in the subscriber's office or may be placed via the PSTN 50 directly to a telephone 76 on the subscriber's desk, or to another telephone where the subscriber is expected to be located, for example, phone 81 in the subscriber's automobile, phone 78 at the subscriber's home 79, phone 75 at factory 74, or to a phone on their person, or wherever the system expects to locate the subscriber. The telephone number of the subscriber's expected location(s) is (are) stored in a memory in the control system 40 and indeed, may be based upon a schedule which tracks the normal movements of the subscriber throughout the day. The schedule may include a "Do Not Disturb" mode in which they would not be paged nor would the system try to contact them directly, but rather, the caller would be connected directly to a voice mail box 30 of the subscriber. As will be discussed with reference to the preferred embodiment, the system can also make more than one call seeking the subscriber and/or to connect the calling party with the voice mail system 90. Such calls can be made simultaneously or sequentially. In the preferred embodiment, the voice mail system is available at a telephone number which is contacted either sequentially, that is, after the other forward leg call(s) are made or concurrently, if the subscriber wants to be able to monitor the incoming call (in which case the incoming call is preferably to be directed to voice mail so that the subscriber can listen to the caller's voice). Alternatively to contacting a voice mail system, the telephone number of the subscriber's secretary, for example, may be stored in

the system so that when the system fails in making a connection with the subscriber within a predetermined time period, the inbound call then would be directed to the subscriber's secretary (for example) instead of to voice mail.

In addition to placing the page and placing the telephone call, the caller is preferably informed that normal call processing is occurring. This may be done by (i) allowing the caller to listen to the sounds generated by the PSTN (or local PBX) while the call is being placed to where the subscriber is expected to be located, or (ii) simulating telephone ringback sounds, or (iii) generating a voice indicating to the caller that the caller has reached the telephone line of the subscriber and the subscriber is being paged to a telephone, or (iv) allowing the caller to interact with the voice mail system 90 if the subscriber wants either to be able to monitor the call or wishes not to be disturbed, or (v) playing music, or any combination of the forgoing, starting at step 22. Typically a timer would also be set for the subscriber to either answer their phone or to answer their page and telephone the system, but that timer is not needed if the caller is sent directly to voice mail.

A test is made at step 24 to determine whether or not the subscriber has answered the page and telephoned the system. The subscriber may use any telephone interconnected to the system to respond to the page, including a pay telephone 72 on the street. If the subscriber has answered the page and telephoned the system, a branch is made to block 16 for which the processing has been previously described. Otherwise, the

processing falls through to another test at step 26 to determine whether or not the telephone number(s), which was (were) called at step 20, has (have) been answered (the voice mail number need not necessarily be tested for this purpose - the purpose here is to see if a human has answered the phone, and, if so, to give the human certain options). If desired, a message may be played for the answeree, advising them that the subscriber has an incoming call, and thus giving the answeree the opportunity to receive the call, reject the call, monitor the call or transfer the incoming call to another network address. At this point the processing preferably falls through to step 36. The two steps discussed above, namely advising the answeree of the call and allowing them to reject it, may not be embodied at all or only partially or may be disabled by suitable programming so that these additional steps might be utilized in some situations and not in others. For example, if the call placed at step 20 is directed to the subscriber's telephone at their desk, such additional steps may not be warranted. On the other hand, when the telephone number which is called at step 20 is a telephone number which is not necessarily primarily and exclusively associated with the subscriber, such as a home telephone number, such additional steps might be included so that the answeree is advised that an incoming call is for the subscriber, giving them the opportunity to accept or reject the call. The timer examines how long the caller has been waiting for the subscriber to either answer a phone or to answer the page, and if the subscriber does not respond within a reasonable period of time, the processing falls through at step 28 so that the caller is then conveyed to a voice mailbox, at block 30, where

they would be invited to leave a message for the subscriber. Of course, this step is unnecessary if the calling party has already been conveyed to voice mail to give the subscriber an opportunity to monitor the call.

Turning now to Box 36, a telephone call has been placed at step 20, and if the telephone call has been answered, the caller is connected to the person who answered the telephone at step 36. This connection may be a full duplex connection or a half duplex connection depending on how the system is programmed for that particular subscriber.

The person answering the telephone can park the call at step 38, assuming they enter the telephone call if the system is set up to allow the party on the forward leg to monitor an incoming call. This park feature is useful because the person answering the telephone may tell the caller that the subscriber will be with them in a few moments and then place the call on park at step 41. As will be seen, in the embodiment disclosed with reference to Figures 4a - 4k, the call can be parked merely by placing the answered phone on hook. At step 42, a test is made to determine if the subscriber has answered the page by phoning into the system. If so, a branch is made to block 16 where the subscriber is placed into telephone communication with the caller, either in full duplex mode or in half duplex mode, as previously discussed. The party on the forwarded leg is preferably also allowed into the communication with the subscriber and calling party and the subscriber is preferably provided with a mechanism to disengage the party (parties) on the forward leg (legs), if desired. If the subscriber has not yet

telephoned the system, the processing falls through to step 44 where, if the call has been parked, a determination is made to see if a park timer has expired. If the caller has been parked for more than a predetermined amount of time, as indicated by the timer, the processing will fall through step 44, and the caller is directed to the subscriber's voice mail at block 30 where the caller can leave a voice mail message for the subscriber. Of course, this could be a second trip to voice mail. For example, if the forward leg calls placed at step 20 are set up in monitoring mode, then the calling party is connected to voice mail at that time and voice mail is paused once the answeree on the forward leg barges into the call. If the answeree exchanges some pleasantries with the calling party and then parks them in anticipation that the subscriber will answer on another forward leg or will telephone the system in response to the page, the timer may expire sending the calling party back to voice mail. Alternatively, the calling party may be advised that they have the opportunity of transferring to voice mail if they tire of waiting for the subscriber, in which case a transfer to voice mail occurs in response to a command entered by the calling party. If the voice mail system has simply been paused, then it would be preferable to restart the voice mail recording process previously initiated as opposed to starting a new voice mail session.

When the subscriber answers the page by telephoning the system, and the call made at step 20 has also been answered, then a three way conference ensues. An appropriate warning tone or message can be played so that the persons in telephone communication realize that

the subscriber is barging into the telephone call. Preferably, the subscriber is provided with a means of disconnecting the party who answered the call on the forward leg to ensure privacy.

It is to be noted that a test is made at both steps 24 and 42 as to whether or not the subscriber has answered the page and telephoned the system. Preferably, that testing would occur throughout the processing described above, and not simply at the places indicated. A more robust version of this system will be described with reference to Figures 4a - 4k, which employs multitasking so that multiple processes can be performed simultaneously.

Turning now to an incoming call being detected from a subscriber, a test is made at step 46 to determine whether or not a call is waiting, i.e., whether there is a caller waiting to be connected to the subscriber. If so, the subscriber may be advised that they have a call waiting by an appropriate message played before the connection is made at step 16. They may also be given the choice of entering the connection in monitoring mode (i.e., in half duplex mode) or entering the connection in full duplex mode).The subscriber may also be advised, at this step, when an additional party enters the connection in full duplex mode, such as the party answering a phone call made at step 20 chooses to connect in full duplex mode.

Of course, the test made at step 46 need not be accomplished if the call meeting process described above is the only process which is performed by the system. Otherwise, when a subscriber calls into the system, and

they have no call waiting, the processing falls through to block 50 where other functions may be performed, such as programming the system (for example to change the mode of operation, to change forwarding numbers, to change a schedule, etc.) or to give access to the subscriber's voice mailbox, etc. The call waiting test made at step 46 would preferably be done on a pre-emptive basis so that even if the caller were in their voice mailbox and if a call were to come into the system from a caller, the subscriber's presence on the system would be detected at step 14 and the subscriber would be advised that they have call waiting. The subscriber and the caller can then be conveyed to a telephone meeting at block 16. Preferably, the subscriber would be first given the opportunity of meeting the call or conveying the incoming call directly to voice mail at block 30 in the event the subscriber wishes to continue his or her activities at block 47.

The foregoing description has been in the context of implementing the invention in a telephone control system 40 of the type depicted in Figure 1.

Turning to Figure 3, this figure shows the system 40 in which the present invention may be implemented and how it interacts with callers, subscribers and other persons. A call is initiated by a caller 2 who places a telephone call, designated by line 3, to system 40. The system responds by (1) paging the subscriber and (2) attempting to telephone the subscriber. Here the paging system is shown at numeral 4 and it is connected to system 40 by a communication facility 5 (which may be provided by the PSTN or by a dedicated channel). The paging system 4 transmits a paging signal 5' to the

subscriber who carries a paging device on his or her person. The subscriber may telephone the system 40 from any telephone 6 interconnected to the system to barge into the call with the caller 2. This connection is called the barge in leg and is designated by the numeral 7. This leg of the call would typically be made via the PSTN. The system in addition to causing a page to occur also tries to contact the subscriber at a telephone 8 having a predefined telephone number. This connection is called a forwarding leg 9 and it may be made via a PBX with which the system 40 cooperates or via the PSTN. The system may initiate more than one forwarding leg (as is represented here by telephone 8' and forwarding leg 9') if it is desired to ring more than one telephone simultaneously with the placing of the page. For example, the system may be programmed to place forwarding leg calls to the subscriber desk telephone and to their car telephone during normal working hours whenever an incoming call arrives in the system 40 from a caller 2 for the subscriber, in addition to placing a page to the subscriber. Once the subscriber is placed in telephone contact with the caller any unused legs may be then torn down or they may timeout of their own accord. When multiple forward leg calls are initiated, the system preferably produces simulated ring back tones or other call progress signals in lieu of providing multiple PSTN call processing sounds which may be confusing to the caller (for example, if two forward legs calls were placed and one number was busy while the other number was ringing, the sounds produced by the PSTN would be rather confusing -- to say the least -- and the calling party should preferably be isolated from those sounds).

If more than one forward leg call is initiated, ringing continues on each forward leg until the associated telephone is answered. Thus if forward leg calls are made to a subscriber's cellular phone and to their office phone simultaneously, the cellular phone continues to ring even if the office phone is answered. Likewise, the office phone continues to ring even if the cellular phone is answered. Of course, the system can be modified so that one phone stops ringing when other is answered. As a further modification, the determination of whether one phone stops ringing when another is answered could be made programmable and perhaps even a function of which phone answers first. Thus, in the foregoing example it might be desirable to have the office phone stop ringing if the cellular phone is answered, but not the other way around (i.e., the cellular phone would continue to ring irrespective of whether or not the office phone is answered).

In the foregoing example, if the office phone is answered, the cellular phone continues to ring. And it continues to ring even if the office phone is hung up. Thus a person in the subscriber's office can speak with the caller and tell them to hold on - the subscriber should be with them in a moment. When the subscriber eventually answers the cellular phone (or answers a page, if they are using a pager), they can speak with the caller who has effectively been placed on hold merely by the action of the person in the subscriber's office hanging up.

A more detailed embodiment of the invention is now described with reference to Figures 4a - 4k. The invention, as described with reference to these figures,

can be implemented in the computerized switch of a telephone switching office or can be implemented in an adjunct computer which controls the switch 41 in a telephone switching office or can be implemented in a telephone control system 40 which is attached to or part of a PBX, for example, in a business. As will be seen, the invention as disclosed in this embodiment has many of the same features as the invention disclosed with reference to Figure 2, but it has several additional features. For example, methods for handling long distance or toll calls are established so that the caller is not charged for a toll call until such time as the caller's telephone call is answered either by the subscriber or by another person who picks up the telephone 8 or 8' on the forwarding leg. Of course, that is the typical way in which toll calls are charged. In the invention as disclosed with respect to Figure 2, when a telephone call is received at step 10, answer supervision might then go back to the telephone company. Answer supervision not only tells the telephone company to start charging for the call but some long distance companies typically disable the voice channel from the caller until answer supervision is returned. The forward channel may be desired in some embodiments, since if no one answers the call, the caller may want to select voice mail, and the system would then need to hear the keys depressed by the caller in order to make that selection. Typically there is no need to enable the forward voice channel from the caller 2 until they are placed in contact with (i) the subscriber, (ii) a party who answered the forwarding leg call, or (iii) the voice mail system (so that the caller can leave a message and so that the subscriber can monitor the call if they so desire). Thus, if the call at that step 10 were a long

distance call, a toll would be charged to the caller, even though they had not yet been connected to the subscriber (or anyone else), if answer supervision were given at that time. The invention, as it will now be described with reference to Figures 4a - 4k, overcomes this and other drawbacks and adds additional features, as will become clear.

The software listing which is appended hereto as Appendices I - X generally corresponds to the flow diagrams depicted in Figures 4a - 4k. While the software listing implements the flow diagrams in most aspects, there are some differences. For example, the description of the flow diagrams indicates handling more than one forward leg call, while the software is currently set up only to handle one forward leg call. Further, the software listing does not include the memo record feature or the music on hold feature, as yet. Such not included features can be readily implemented by those skilled in the art.

The software is written in a language which is specifically adapted to test telephone system technology and is called VOS (Voice Operating System). The software is available from Parity Software Development Corp., of San Francisco, CA and runs on the hardware environment of system 40 depicted in Figure 5, which will now be described. The hardware environment of the system 40 includes an IBM compatible PC (personal computer) 90 having CPU of the Intel 486 family or better. Preferably, the computer is an AST Bravo MT 486DX2 66 with 16 Meg of RAM, a large SCSI hard drive, two serial ports, one parallel port, mouse, video and DOS, or equal. The VOS software supports limited multitasking.

The flow diagrams, which will now be described, assume that the system 40 runs in a multitasking environment and those skilled in the art will appreciate in comparing the flow diagrams with the computer code listing that certain compromises were made in implementing the flow diagrams into code.

The hardware platform also preferably includes a Dianatel SmartSwitch96 with SmartBridge96 (switch serves as switch 41 in system 40), Dialogic LSI/120 interface cards, and Dialogic D/121 voice cards. These cards plug into the computer motherboard and are also connected to each other via PCM data busses, as shown in Figure 5. The software listing in the appendices and the Dialogic cards identified above and in Figure 5 are set up for working with POTS (Plain Old Telephone Service) loop start lines. With small adaptations to the code and by selecting different interface cards (also available from Dialogic), then T-1 trunks can be used instead. Figure 6 shows the hardware platform with Dialogic cards installed in the PC for use with trunks 51. In either case, this hardware environment can only support a limited number of active incoming calls. This is fine for a typical business application, such as where the hardware controls a PBX installed at the business, but if this system is to be scaled up to work with large numbers of callers, such as at a telephone switching office, then a more robust hardware/software environment should be used. Those skilled in the art will appreciate that several PCs can be used in tandem and certainly more powerful CPUs can be utilized. VOS was selected as the development environment for the software, but those skilled in the art may choose to use another language in which to implement the invention. The computer may be

used to generate sounds, especially if a sound board is added. For example, a SoundBlaster board can be used to generate sounds and computer generated speech as well as using known boards of the SoundBlaster type. The computer may be equipped with a sound generation board if it is desired to generate call announcements over speakers 71 (Figure 1). Thus the signaling system may be provided by a conventional pager system or a digitized speech system which delivers audible communications over speakers 71 or by a combination of the two systems. Paging could alternatively be accomplished by sending signals to computers on the desks of subscribers, which computers could then not only provide the desired paging function, but could also advise the subscribers of the identities of callers by making use the calling party's CLID. Additionally the Dialogic D/121 voice cards can produce digitized voices stored, for example, on the hard drive, in telephone connections with callers 2, subscribers 65 or parties 8.8' on the forwarding legs. Thus a means for producing courtesy messages to users of the system 40 is available.

Additionally, the system may be connected to receive and send Signaling System Seven (SS-7) messages from and to the telephone company. SS-7 is well known in the telephone industry, and therefore does not need to be described here. Figure 5 depicts an optional SS-7 messaging translator which can be embodied in a R-5 computer by Stratus with SINAP option.

The main task 100 is shown in Figure 4a. The main task 100 starts off by a step 102 in which the variables which are used during the processing are set to their default values, typical initialization routines run,

drivers loaded, etc. After the variables are set up and other housekeeping chores accomplished, the processing falls through a loop which is made around a test made at step 104 to determine if a new inbound call has been received. The inbound call could be a call 3 from a caller 2 or a barge-in or meet-me call 7 from the subscriber or a call from the subscriber merely to check their voice mail, to update one or more forwarding numbers stored in the system or a schedule of forwarding numbers, etc. If a new inbound call has been detected, the processing falls through first to step 106 where the inbound channel is determined, then to a step 108 where the Calling Line IDentification (CLID) is obtained (if available) and then processing then falls through to step 110 where the Inbound Task 200 is started to process the call on the indicated channel.

The Inbound Task 200 is shown in Figures 4b and 4b(1) - 4b(7). The Inbound Task 200 starts off at Figure 4b by obtaining the Direct Inward Dial (DID) number at step 201. In this embodiment, it is assumed that the subscriber uses one telephone number to call the system, while the subscriber's callers use a different telephone number to call the system. Therefore, the subscriber and the caller can be differentiated by the DID digits. Those skilled in the art appreciate, of course, that the DID digits are supplied by the telephone company to the system. At step 202, a test is made to determine whether the DID digits correspond to the telephone number used by one of a plurality of subscribers in order to gain access to the system. If the DID digits indicate that one of the subscribers has called in, then the processing falls through to step 235, which is continued in Figure 4b(4). Otherwise, a test is made at step 203

to determine if the DID digits indicate that a caller is trying to telephone a subscriber, in which case the processing branches via connector 205 to the processing shown on Figure 4b(1). If the DID digits do not result in a positive test at either step 202 or 203, the processing falls through to step 204 where the call is disposed of and the task is then suspended at step 204.1.

By way of a practical example, subscriber 1 may have a telephone number 555-7000 for use by callers and a telephone number 555-7001 for use by the subscriber to gain access. A second caller may have telephone numbers 555-7010 for use by callers and 555-7013 for use by the subscriber to gain access. Of course, it would be envisioned that many more subscribers would be on the system than in this small example, but in this example, the valid DID digits would be 7000, 7001, 7010 and 7013. If a call were made to 555-7009, the processing would fall through to step 204 where the call would be disposed of. That could be by, for example, playing a message to the caller indicating that they had reached an unassigned telephone number. If the DID digits indicate that a call were made to 555-7013, then that call would be identified as a call from the second subscriber for the purpose of gaining access to the system (e.g. to barge into an existing call or to monitor an existing call or to perform other functions). Similarly, if the DID digits indicate that a call were made to 555-7000, that call would be identified as a call from a caller directed to the first subscriber.

Assuming that the caller dialed a valid caller access DID, such as 555-7000 for the first subscriber on

the system, the processing then falls through via connector 205 to block 206 on Figure 4b(1) where a test is made at block 206 to determine if the monitor mode is enabled. In the monitor mode the subscriber can monitor the caller(s) to determine if the caller wants to barge into the conference (the caller can be in communication with either the voice mail system or with a person answering on a forward leg) where the caller has been placed by the system. As the reader will note, two different processing paths are taken if the monitor mode is active. Many of the processes in each path have identical or similar processes in the other path. Where such process are identical, or are at least very similar, the same reference numeral is used, but in the monitor mode path those reference numerals have the letter 'M' appended thereto. For example, after step 206, in each path the 'caller' variable is set active for that particular subscriber at steps 207 and 207M. As will be seen, these two paths accomplish very similar functions, and, indeed, the two paths could be combined as the difference in the functions are not highly significant. Rather, they demonstrate two slightly different methods of handling an inbound call by the system each of which have certain advantages.

Considering first the monitoring path, processing then continues on to the next block 208M where the Caller Disconnect Task 900 is started. The Caller Disconnect Task is described in detail with reference to Figure 4i. After the Caller Disconnect Task has been started, the processing then drops to step 209M where a page to the subscriber is initiated by starting the Start Paging Task 300 (which is described with reference to Figure 4c), and on to step 210M where a variable

called "call waiting" is set true for the subscriber.

At step 212M a test is made to determine if the forwarding number, i.e., the number which is used on a forward leg call, is set to the null character. If so, then no forwarding number is in the system and processing goes on to the Caller Monitor routines via connector 216. If a forwarding number is in the system, then a test is made at step 213M to see if the forwarding number programmed into the system is the same a the DID number. If so, the variable "four rings played" is set to true at step 214. The reason for this is that the present system is preferably used as an adjunct to, or embodied with, the telephone control system disclosed in U.S.Patent No. 5,375,161. Thus the forwarding number may be set to the subscriber's telephone number used in the apparatus disclosed in that U.S. Patent. If so, then the present system directly forwards the incoming call to that system externally of the PSTN without generating any ringing sounds. Otherwise, such ringing sounds are normally generated, as will be seen.

A Forward Leg Task 500 is started at block 215M and then the processing falls through to the Caller Monitoring routines on Figure 4b(2) via a connector 216. As previously described the system may preferably be programmed to forward the call on a number of forwarding legs simultaneously or in sequence. The system preferably includes a database which stores the various forwarding numbers for a subscriber along with perhaps a schedule indicating at what times the various forwarding numbers are effective. For example, during working hours, including the time the subscriber is normally

commuting to and from work, they may decide to have both their work telephone number and a cellular number stored as their forwarding numbers for that time period. At other times they may have their home telephone number stored as the forwarding number, and at still other times they may have programmed the system to direct calls to the voice mail box, or invoke call screening options. Such personal preferences would be preferably stored in a database associated with the system. At the point the processing reaches the block labeled "Start Forward Leg Task," the Inbound Task would look to the database not only to determine the forwarding number then in effect, but preferably also to determine if multiple forwarding numbers were then in effect. If so, the Forward Leg Task 500 would be repeatedly called at this point, once for each forward leg required.

It is to be recalled that this is a multitasking environment such that the Timer Task, Paging Task, the Forward Leg Tasks, and the Caller Disconnect Task can all be running simultaneously with this particular Inbound Task. Multiple instances of the Forward Leg Task and associated disconnect tasks may also be running. Finally, multiple instances of the Inbound Task are likely to be running together with its progeny, i.e., the tasks it calls either directly or indirectly. After the Forward Leg Task 500 is started at block 215M, the caller is added to a conference facility. In due course, assuming that the subscriber makes contact with the system, the subscriber will also be added to that conference. If multiple callers are on the system at the same time trying to contact different subscribers, then multiple conferences are similarly set up. After step 215M, the processing falls through a connector labeled

with the numeral 216, which can also be found on Figure 4b(2).

Referring now to Figure 4b(2), the processing first enters a first loop comprising steps 217M, 218M, 220M, 222M, 223M, 224M and 225M. At step 217M a test is made to determine if either four rings have occurred or if the variable 'four rings played' is currently set true. If yes, the incoming call branches to a block where a voice mail subroutine 234 is called, before returning to the first loop. This branch is taken if either four rings have been played at step 219M or if the subscriber has called in or answered a forward leg call. In the latter case, the incoming caller is sent immediately to voice mail so that the subscriber can monitor the caller's interplay with voice mail. The processing loops in the first loop until voice mail comes up (a voice mail call is answered if it is, for example, an external voice mail machine), in which case the test made at step 225M causes the first loop to exit via steps 226M and 226.1.

Continuing through the first loop, the "no" branch from step 217M enters a test at step 218M which determines if a ring should be generated. Since the caller has not yet been added to the conference, and since answer supervision has not yet been returned, the ringing sounds are artificially generated. Ringing sounds have a particular cadence, and the test at step 218M assures that the ringing at step 219M follows the desired cadence. If the party enters the ## keys at the keypad of their telephone, that action is detected at step 220M for the purpose of setting the "priority call waiting" variable at step 221M. If the party enters the

*9 keys at the keypad of their telephone, that action is detected at step 222M and the voice mail subroutine 234 is called. If a forward leg has been answered, then the test made at step 223M will cause the processing to exit via its "yes" leg to block 226. If the subscriber has barged into the conference on the barge-in leg 7 and has left the monitoring mode (i.e. depressed the * key as tested at step 243, Figure 4b(4)), then the a test made at step 224M will cause this loop also to be exited to block 226. This might occur after the first loop is exited at step 217M in order to start voice mail and then reentered, but with the test at step 225M failing (because voice mail failed to respond). Thus the caller and the subscriber can still be placed into telephone contact in the monitoring mode even if voice mail fails.

If a forward leg is answered, or if the subscriber barges into the conference, or if the calling party is sent successfully to voice mail, then the processing falls out of the loop to block 226, as previously mentioned. At block 226.1 answer supervision is returned to the caller (i.e. toll charges start if it is a long distance call and two-way communications become available - two way communications are often inhibited by long distance carriers until answer supervision is returned). After block 226.1 a second loop is entered which comprises steps 228M, 257M, 229M, 230M, 231M, 232M and 233M. If the caller enters *9 at the keypad of their telephone in this loop, the loop exits via step 227M where music is stopped (assuming it was started at step 238M), and calls the voice mail routine 234. Thus, the caller enters the first loop where they can hear generated ringing sounds. If by four rings either a forward leg answers or the subscriber has called in (in

response to a page), or four rings have occurred, and the caller has been sent to voice mail and voice mail has answered, then the second loop is entered which normally just loops around steps 228M, 257M and 229M. While in the second loop, if the subscriber answers a forward leg call, or barges into the call in response to a page, then the subscriber is added to the conference with the calling party.

The second loop expands to include steps 230M, 231M, 232M and 233M when the forward leg answers (as detected at step 229M), and then hangs up (as detected at step 230M) so long as the calling party is not active in voice mail. For example, the person answering on a forward leg may tell the caller that the subscriber is expected to phone the system in order to talk with the caller, and therefore the caller should wait on the line. During this waiting time, the caller hears a message about leaving a message (at step 232M), or music (via step 233M). The second loop also tests at step 228M for the caller touching *9 (to go to voice mail via subroutine 234).

If monitoring mode is not active, then the test at step 206 on Figure 4b(1) takes the other branch. The call processing of the inbound call is similar to the monitoring mode processing just described, with the following more significant differences: Answer supervision is returned earlier (at step 207.1 instead of at 226.1), and the caller is added to the conference earlier (at step 211 instead of at step 226). Thus, instead of hearing a simulated number of rings (as at steps 218M and 219M), the caller hears one ring (step 213.1) and messages (at steps 213.2 and 213.3) before

the Forward Leg Task 500 is started at block 215. Additional simulated ringing is played at steps 217 and 218 only if there is no forwarding number (i.e., the forwarding number is equal to null). Typically, processing is via block 213 and the message blocks to block 212 then thence to step 215, and then on to connector 256 (see Figure 4b(3)).

Turning now to Figure 4b(3), the call processing on this sheet is similar to Figure 4b(2) in that there is a first loop (comprising steps 220, 222, 223, 224 and 225), and a second loop (comprising steps 228, 257, 229, 230, 231, 232, and 233). The call handling is nearly the same for both modes (monitoring and non-monitoring), the more major differences having been spelled out above. These differences have little impact on whether the subscriber decides to utilize the monitoring mode, and thus are primarily set forth here as two possible techniques for handling the incoming call from a caller.

Turning now to Figure 4b(4), assuming that the DID digits detected at step 202 were associated with the subscriber barge-in telephone number, such as 555-7001 in the prior example, the processing falls through connector 235 to step 236 where answer supervision is returned for the barger's telephone call. At step 237 a Barger Disconnect Task 400 is started. Processing then continues on to step 238 where a test is made to determine whether or not a caller is active on the system. If not, the processing branches to step 248 after playing a ring at step 238.1. Otherwise, a caller is on the system and the processing falls through to step 239 where a test is made to determine whether or not the subscriber has a Personal Identification Number

(PIN) stored. If a PIN is stored, then the "no" leg is taken and a ring is played at step 239.1. Otherwise, the "yes" leg is taken to step 258 where a test is made to see whether or not the monitoring mode is active. IF the test at step 258 is "yes", then the processing falls through to step 241 where 'four rings played' is set active.

If the "no" leg is followed from step 238, meaning that no caller is on the system, then the party who telephoned is given a short period of time (one second in the present embodiment) at step 248 to start entering a PIN after the ring is played at step 238.1. In this embodiment it is envisioned that the subscriber has two personal identification numbers: one for the purpose of programming (which may be a longer, more complex PIN), and the other to meet a caller in the conference (i.e., to barge into the conference) for which a different PIN, which is preferably shorter, or even nonexistent, may be used to make it easier to enter and to gain access to a telephone call). If the person starts entering data within the one second delay, then processing branches to step 250.

If the "no" leg is followed from step 239, meaning that a caller is active and the subscriber has a PIN which must be entered, a message is played at step 239.2 advising that a call is waiting and a PIN is requested at step 249. If the subscriber has entered their programming PIN, then a test made at step 250 is satisfied, and processing falls through to step 251 wherein the subscriber can change their program variables, their forwarding number(s), voice mail attributes, their schedule, activate or deactivate the

monitoring mode, or gain access to their voice mail system. Those tasks are not described in detail here since it is known in the art how to change such variables.

If the person calling on the barge-in line inputs the barge-in PIN at step 252, it must be the subscriber who is trying to join the conference. Processing then falls through to step 253 where a test is made to see if a caller is active for this subscriber. Normally, if the subscriber is responding to a page, then the caller variable is active, and, if true, processing continues via block 258. If a caller is not active, the subscriber is advised at step 254 that there is no call waiting, and processing returns back to step 249. If the person telephoning does not enter a PIN at step 248, then the processing continues via connector 280 to Figure 4b(6).

The variable "four rings played" is set at step 241. If the calling party is still in the first loop of Figure 4b(2), setting "four rings played" variable to true causes the calling party to be immediately transferred to voice mail where the subscriber can monitor the calling party. Indeed, step 242, which follows step 241, adds the barger to the conference in a listen only mode (i.e. a half duplex mode) about the same time the calling party is added to the conference. As will be seen, the voice mail system is also connected to the conference for the purpose of recording the incoming party's message (and thus allowing the subscriber to ascertain who is telephoning before taking the call).

A short loop comprising steps 243 and 244 is

entered while the subscriber monitors the incoming call. The subscriber can (i), enter the conference by pushing the * key (as tested at step 243); or, (ii), remove himself or herself from the monitoring mode by either depressing the # key (as tested at step 244 and implemented at step 255), or simply hanging up (in which case the subscriber is removed from the conference by the Barger Disconnect Task 400). If the subscriber decides to speak with the calling party, they merely depress the * key and then enter the conference in full duplex communication at step 246. At the same general time, the variable "barger" is set active (see step 245), and, after the subscriber barges in, a joining tone is played at step 247 to advise the calling party that the barger is coming in and call waiting is reset at step 247.1. Processing then proceeds to the Barger Answer routine on Figure 4b(5) via connector 260.

Referring now to Figure 4b(5), at this point the subscriber has been identified as wanting to barge into the telephone call with the caller, and since the caller is in a conference (see steps 211 and 226), the subscriber has been added to that same conference (at step 246), and a tone was played (at step 247) which all parties in the conference can hear so that they know that the subscriber has barged into the call. The processing on the barger leg 7 now enters a loop at steps 290, 262, 266, 268, 269, 270, 273, and 275 to determine whether the subscriber 65 enters certain codes at his or her keypad of their telephone instrument 6 and to perform certain other tests. As will be seen, certain codes control a call waiting feature, while other codes allow a message to be recorded, or an outbound call to be initiated, or forward leg calls to be brought down,

or other features to be utilized.

In the example shown in Figure 4b(5), the loop first calls a subroutine 290 (see Figure 4b(6)) which checks to see if a new call from another caller is received. The loop next tests at step 262 to determine if ** is entered at the subscriber's keypad. If so, the subscriber can first answer the subsequent call and then toggle between two conferences at steps 263 and 265. The loop then tests for #8, or #9, to be entered at the keypad. If #8 is entered (as tested at step 266), that will stop forwarding leg tasks at step 267, and as will be seen, that action will disconnect the forwarding legs 9, 9' if it has (they have) been connected. If the subscriber enters #9 (as is tested at step 268), the voice mail subroutine 234 is called. If more than one caller have telephoned the subscriber at more or less the same time, they will join separate conferences and the variable "call waiting" will be set active (i.e. true).

The loop next tests (at step 269) for #4 being entered at the keypad indicating that the subscriber wants to initiate an outbound call. If so, an Outbound Call subroutine 277 is called. The loop next tests (at step 273) for #3 being entered at the keypad indicating that the two conferences are to be joined together. A test is made at step 271 to determine if two conferences are active, and, if so, then they are joined at step 272 and an appropriate tone is played at step 272.1. The loop also tests for the entry of #6 (at step 273) and #7 (at step 275) for starting and stopping a recording device. This feature allows a party to the conference to take a voice memo and store it, preferably as a voice

mail message for the subscriber. Thus, important topics which are discussed with a caller can be recorded. If needed, a periodic tone can be easily generated while the parties conversation is being recorded. The recording task is started at step 276 and stopped at step 278. This loop has no explicit exit, but it is exited when the Inbound Task for the barger is suspended by the Barger Disconnect Task 400.

One of the advantageous features of the present invention is that either the subscriber, or the caller, or any party on a forwarding leg, can start the voice mail task 700 when they are in telephone communication via the conference with another party so that a joint message may be left in the subscriber's voice mailbox. This can be very useful in certain situations. For example, the subscriber may be in an automobile and may be connected to the system via a cellular telephone. That can be a very inconvenient time, indeed, an unsafe time, to take notes. By allowing the voice mail to be started while the parties are in the conference, by any party, it allows a voice note to be stored which the subscriber can then go back and listen to when in a more convenient location. As has been described, the system also permits the caller to start the voice mail task 700 (see steps 222 and 222M) so that the subscriber need not take his or her hands off the automobile wheel in order to have the voice mail task 700 started.

Another advantageous feature of the present invention is that the subscriber can make outbound calls in a new conference while speaking with a party in another conference. For example, the subscriber, while speaking with a first party, may decide that they need

more information from a second party. The subscriber can call the second party in a new conference, speak with them, and then either toggle between the two conferences (by keying **), or join the two conferences into a single conference (by keying #3). The Outbound Call subroutine 277 on Figure 4b(6) implements this feature.

In the foregoing description, the various loops wait for keys to be pressed in order to invoke additional features. Voice recognition systems are well known, and therefore some practicing the present invention may find it useful to have the loops sense not only for the operation of keys on a keypad, but also to initiate the described functions in response to voice commands.

Turning now to Figure 4b(6), the "Barger NoEntry" routine 280, "Start Voice Mail" subroutine 234 and "Barger Call Waiting" subroutine 290 are depicted.

In the Barger NoEntry call processing, the caller on the barge-in leg has not entered a PIN as tested at step 248 on Figure 4b(4) and, of course, no call was waiting to be answered. The party is permitted to enter voice mail, in that a new conference is set up at step 281, and voice mail is started at step 282, and added to that conference. The Barger Call Waiting routine 290 is called followed by step 284 and 285 which loop with the call to routine 290. In this loop, a test is made at step 284 for the entry of "***" at the keypad of the barger's telephone instrument. If a caller calls in while the barger is in active invoice mail, the barger will hear two beeps (played by routine 290) and if the barger enters ** at the keypad, barger is set active at

step 286 before branching to step 287 (the barger toggle connector).

The Barger Outbound Call routine 277 is also depicted on this figure. At step 277.1 a test is made to determine if a call destination is selected. The "no" leg is showing looping on step 277.1, but it is understood that if a valid destination is not selected after reasonable opportunity is given to do so, or if the barger elects to escape from this function, the routine would simply return to its calling point. If a valid destination is selected, then, if necessary, the destination phone number is looked up at step 277.2 (for example, a speed dial number may be used to identify a destination) before an outbound channel is seized (step 277.3) and the number of the destination is dialed (at step 277.4). The barger is added to a new conference (step 277.5) and the outbound channel is added there as well (step 277.6). If the outbound call is answered, as is tested at step 277.7, the routine simply returns at step 277.11. Otherwise, if busy, or no answer occurs, then the barger is switched back to the conference from which he came at step 277.8, the outbound call is disconnected from the conference, and the call is torn down at steps 277.9, and 277.10.

Turning now to the Barger Call Waiting subroutine 290, if call waiting is not active (as tested at step 290.1), the routine merely returns. Otherwise call waiting tones are played twice (as tested at step 290.2) to the subscriber. For normal call waiting, one type of waiting tone is played (at step 290.6), while if the caller has indicated that their call is a priority call (see steps 220, 221, 220M and 221M), a different call

waiting tone is played at step 290.5.

The "Start Voice Mail" subroutine 234, which is depicted on Figure 4b(7), starts off by testing to see if voice mail is already active at step 234.1. If so, nothing further is needed and the subroutine exits at step 234.7. Otherwise, a test is made at step 234.2 to determine if voice mail is enabled. If not, a courtesy message is played at step 234.3 and the routine exits (returns) at step 234.5. If voice mail is enabled, then the Start Voice Mail Task 700 is called at step 234.4 before exiting at step 234.5.

In this embodiment it is assumed that voice mail is provided by a separate service provider available via the PSTN 50 (such a service provider may have a voice mail equipment of the type made by Octel, for example). Thus, some subscribers may opt not to subscribe to a voice mail service and that choice would be stored in the aforementioned database (together with the telephone number of the voice mail service, if available for the subscriber). Thus, the test made at block 234.2 can be determined by examining the database, for example. If voice mail is available, the Voice Mail task 700 is started if it is idle for this subscriber at step 234.4. Similarly, if voice mail is not available, the caller would be preferably informed of that fact at block 234.5, after which the subroutine returns at 234.5.

Instead of utilizing an external voice mail system, the voice mail system may be embodied with the present system, if so desired.

Turning now to the Paging Task 300, that will now be described with reference to Figure 4c. Recall that the Paging Task runs concurrently with the Inbound Task (see steps 208 and 208M on Figure 4b(1)). When the Paging Task is started, an outbound channel is seized at step 302. The telephone number of the subscriber's paging system is dialed at step 304. Appropriate I.D. information is output to the pager at step 306 - this will preferably include the CLID of the caller (if available) so that the pager can display the telephone number or other information about the caller on the display of the pager. The CLID, if available, can also be used to look up information about the caller, such as the caller's name, etc., in the database for transmission to the subscriber. Thereafter, the paging channel can be disconnected at step 308 and the task suspended at step 308.1. This processing assumes, of course, that the pager is controlled by a typical dial-up pager connected via the PSTN 50. That is a common way of dealing with the pager, since the pager system facilities are often provided by a separate company. However, another way of dealing with the pager system is to use a direct, dedicated connection between system 40 and the pager system, so that instead of seizing an outbound channel and dialing the pager company via the PSTN, such as shown in steps 302 and 304, the system would preferably seize the dedicated connection to the pager company and output the necessary information directly at step 306. Additionally, the Pager Task can also be used to instruct a digitized speech system to deliver audible pages over speakers 71. Indeed, the Inbound Task 200 can start more than one instance of the Pager Task 300 so that the subscriber is paged both audibly and via a conventional pager device

61, if desired. At the same time the Paging Task(s) 300 is (are) being run, the Forward Leg Task(s) 500 is (are) also being run.

The Barger Disconnect Task 400 is shown at Figure 4d. A test is made at step 402 to see if a disconnect which has occurred on the barge in leg 7. If so, the barger variable is set inactive (false) at step 404 and the call waiting variable is similarly set inactive (false) at step 406. The subscriber is disconnected from the conference at steps 408 and 410. The caller and the party on an active forward leg will remain in the conference. This is a useful feature. The subscriber may barge into a conference and find their secretary and the caller already engaged in a conversation. The subscriber's secretary, in this example, is on a forward leg 9, 9'. If the secretary can handle the matter for the caller, the subscriber can simply excuse themselves from the conference and hang up - the caller and the secretary on the forward leg remain in communication. Before suspending the Barger Disconnect Task at step 414, the Inbound Task 200 which the subscriber (barger) is on is suspended at step 412.

The Forward Leg Task 500 will now be described with reference to Figures 4e(1) - 4e(4). The forwarding active variable is set at step 502 and then an idle outbound channel is seized at step 504. The number of the telephone 8 on the forwarding leg 9 is then dialed at step 506. This may be a number available via the PSTN or may be an extension number of a directly connected PBX. The forward leg channel is then added to the conference at step 508 so that the calling party can hear the call progressing (alternatively they can be

supplied with generated sounds which may sound like normal call processing, such as the rings generated in the first loop on Figure 4b(2)). If multiple legs are being utilized, then it would probably be preferable to either supply the caller with generated sounds or to delay adding a forward leg to the conference until the forward leg is answered, otherwise hearing multiple legs being dialed and/or ringing, more or less simultaneously, would not be especially helpful to the calling party. In that case, or alternatively, a message could be played to the caller informing that the system is attempting to contact the subscriber at a number of locations.

The Forward Leg Disconnect task 600 is then started at step 510, and, thereafter, the processing proceeds to step 512 where a first timer is started, and then to step 514, where a test is made to determine whether the forward leg call has been answered. If so, the processing then continues to the forward routine on Figure 4e(2) via connector 540. Otherwise, a test is made at step 518 to determine whether the forwarding leg is busy and, if not, then a test is made at step 520 to determine whether the first timer set at step 512 has expired. If the first timer has not timed out, then program loops, waiting either for the forward call to answer or for the first timer to timeout. If the first timer times out, or if a busy condition is detected, then a call to the voice mail subroutine 234 is made before falling through to step 522, where a test is made to determine if voice mail is available. If so, a second timer is set at step 528 and answering of the forward leg is again tested at step 530 in a small loop with block 532 where expiration of the second timer is

tested. If it expires, or if the test made at step 522 fails, then forwarding is set inactive at block 534 to remove this forward leg from the conference and to disconnect this forward leg channel. The Forward Leg Task is thereafter suspended at step 536. Otherwise, if the forward leg is answered as tested at step 530, the processing goes to the forward routine on Figure 4e(2) via connector 540.

A busy condition can be tested by a number of different techniques. For example, the system can be listening for the tell-tale busy sound. The Dialogic card previously mentioned have such capability. If the system is connected to the PSTN via SS-7, then a data packet can be returned to the system indicating busy without even the need to open an audio channel to the number being telephoned. Finally, when the system is connected to or with a PBX, the PBX can readily advise whether or not an extension is busy.

Assuming that the forwarding leg call is answered, as is tested at steps 514 and 530, the processing continues via connector 540 to the flow diagram of Figure 4e(2) and to block 541 where a test is performed to see if monitoring mode has been enabled. If not, further processing is through the forward answer routine on Figure 4e(3) via connector 560, after first adding the forward leg to the conference as a listener (i.e. in full duplex mode) at step 547, stopping the voice mail task (at step 548), and testing to determine if the barger is active (at step 549). If the barger is not active, then call waiting is also reset at step 550.

If monitoring mode has been turned on, for example

by the subscriber, then the subscriber is first added to the conference as a listener (i.e., in half duplex) at step 542 before a loop comprising blocks 543 and 544 is entered. In the loop the subscriber can listen to the caller's interplay with voice mail and make a decision regarding whether or not to join the conference which the caller is in. The subscriber joins by depressing the * key (as tested at step 543), and leaves by depressing the # key (as tested at step 544), or, alternatively, by hanging up. If the subscriber exits by the * key, then they are removed from the conference at step 551, the forward channel is disconnected at step 552 and call forwarding is set inactive at step 553. The task then suspends at step 554. If the subscriber depresses the * key, then the processing continues via block 546 where a joining tone is played before continuing with steps 547, 548 and 549.

If the subscriber is going to barge into the call, then processing continues on Figure 4e(3). The processing on the figure is very similar to the processing previously described with reference to Figure 4b(5). There is good reason for this. The processing on Figure 4b(5) tests for commands from the subscriber to enable certain features. The processing on Figure 4e(3) test for many of the same commands for the party answering a forward leg call. Thus, the processing loops in a loop which includes a call to a Forward Leg Call Waiting subroutine 590 and steps 562, 566, 568, 569, 570, 573, 575 and 578.

The party on the forward leg can toggle between conferences as tested at steps 562, 562.1, and 563. Compared with the corresponding testing made at steps

262 and 263, the testing on the forward leg includes an additional test (at step 562.1). The additional test at step 562.1 tests for the barger being active. If so, the ability to toggle between conferences is taken away from the person on the forward leg.

Similarly at step 570.1 where the ability to join conferences by depressing the #3 keys is taken away from the person on the forward leg if the barger is active. Otherwise the processing of steps 566, 567, 568, 569, 570, 571, 572, 572.1, 573, 574, 575 and 576 corresponds with the previously described processing of similarly numbered steps 266, 267, 268, 269, 270, 271, 272, 272.1, 273, 274, 275 and 276, respectively. The loop on Figure 4e(3) also contains an explicit exit via step 578 which is described below.

A test is made at 568 to see if the person on the forward leg enters #9 at their keypad. That person would typically be the subscriber, but could be a third party who answered the caller's telephone call. If they enter #9, the Voice Mail subroutine 234 is called. The test at step 578 determines whether the caller or barger is active. If neither the caller 2 nor subscriber on the barge-in leg 7 is active, there is no point in allowing the forward leg to remain up, so the processing falls through to step 579 where the forwarding is set inactive, and thereafter, the forwarding task is suspended at step 579.1.

The Forward Outbound Call routine 577 is depicted on Figure 4e(4). At step 577.1 a test is made to determine if a call destination is selected. The "no" leg is showing looping on step 577.1, but it is

understood that if a valid destination is not selected after reasonable opportunity is given to do so or if the subscriber elects to escape from this function, the routine would simply return to its calling point. If a valid destination is selected, then, if necessary, the destination phone number is looked up at step 577.2 (for example, a speed dial number may be used to identify a destination) before an outbound channel is seized (step 577.3) and the number of the destination is dialed (at step 577.4). The person making the outbound call is added to a new conference (step 577.5) and the outbound channel is added there as well (step 577.6). If the outbound call is answered, as is tested at step 577.7, the routine simply returns at step 577.11. Otherwise, if busy, or no answer occurs, then the party is switched back to the conference from which they came at step 577.8, the outbound call is disconnected from the conference, and the call is torn down at steps 577.9 and 577.10.

The Forward Call Waiting subroutine 590 is also depicted in this figure. If the barger is active (as tested at step 591), or if call waiting is not active (as tested at step 590.1), the routine merely returns. Otherwise call waiting tones are played twice (as tested at step 590.2) to the party. For normal call waiting, one type of waiting tone is played (at step 590.6), while if the caller has indicated that their call is a priority call (see steps 220, 221, 220M and 221M), a different call waiting tone is played at step 590.5.

Turning now to Figure 4f, the Forward Leg Disconnect task 600 is now described. This task bears some resemblance to the Barger Disconnect Task 400

previously described. Here, an initial loop comprises two tests, made at steps 602 and 604, instead of one test. The test made at step 602 detects whether a disconnect has occurred on the forward leg which this task is monitoring. The test made at step 604 tests a variable "forwarding active" to see whether forwarding is active. If it has been set inactive, then the forward leg is brought down, i.e., disconnected. This can occur, for example, under control of the subscriber on the barge-in leg, if the subscriber on the barge-in leg satisfies the test at step 266, for example, by depressing the keys #8, in which case forwarding leg active variable is reset (set inactive). In that way, the subscriber on the barge-in leg can cause the forwarding leg to disconnect. That can be useful where the subscriber wants to make sure they have a private conversation with a caller, without fear of having someone eavesdropping on a forward leg 9, 9'. Otherwise, if a party is on the forward leg, the subscriber is on the barge-in leg, and the caller is on the caller leg, a three-way (or more) conference will be in place.

After the loop is exited, forwarding is set inactive at step 606 and the leg is removed from the channel at 608, disconnected at 610, and the associated Forward Leg Task 500 is suspended at 612. Then this task suspends itself at 614.

As previously discussed, Voice Mail Task 700 is preferably provided which can attach the members of the conference to the subscriber's voice mail system so that notes may be taken or so that a message for the subscriber may be recorded. The voice mail task 700 is described now with reference to Figure 4g.

The voice mail system may be either a separate stand alone system, for example of the type disclosed in U.S. Patent No. 5,375,161 (the disclosure of which is hereby incorporated herein by reference), or an integral system. In the present embodiment, it is assumed that the present system is connected to the Telephone Call handling system disclosed in U.S. Patent No. 5,375,161 by dedicated connections. However, connections via the PSTN 50 may also be used. In either event, the voice mail system is controlled by inputting appropriate codes into the voice mail system. Assuming for the moment that the voice mail system is a separate stand-alone system, the voice mail variable is set active at step 702, an output channel is seized at step 704, the voice mail telephone number is dialed at step 706 and appropriate codes to access the subscriber's voice mailbox are outputted. The codes may be stored in the previously described database. Thereafter, at step 708, the voice mail channel is added to the conference, and the voice mail disconnect task 800 is then started at 710. Next, a voice mail timer is started at block 712 to ensure that the voice mailbox is answered within a predetermined time period. That test is made by looping at steps 714 and 716. If the timer expires, the voice mail variable is set inactive at step 718 and the task suspends at step 722. If the voice mail system answers, as is tested at step 714, the processing goes through to a short loop at step 720 where a test is made to see if the caller or the barger is active. If neither one are active, the processing falls through to step 718 where voice mail is set inactive and the task is then suspended at step 722.

The Voice Mail Disconnect task 800 is shown in

Figure 4h. A test is made at step 802 to see if a disconnect has occurred on the voice mail leg. Normally, one would not expect a voice mail system to disconnect, but that could occur, and therefore, a test is made for that event at step 802. More likely, some action will set voice mail inactive, in which case the processing will fall through from the test made at step 804 to step 806, 808 and 810. At step 806, voice mail set inactive, if not already done so by some other action. The voice mail channel is removed from the conference at step 808, the voice mail is disconnected at step 810, the voice mailbox task is suspended at step 812, and the voice mail disconnect task suspends itself at step 814.

The Caller Disconnect Task 900 is shown in Figure 4i. This figure is very similar to Figure 4f, and therefore, it is not described in detail other than it is noted instead of checking a forward leg 9, 9', the caller leg 3 is checked. If the caller active variable becomes inactive, the caller link is brought down. That could occur, for example, by the subscriber entering a defined key sequence while in the loop depicted on Figure 4b(5). Of course, that would require another test in that loop to detect the defined key sequence, and then the subscriber would have the ability to cause all connections to come down.

The Timer Task 1000 is depicted in Figure 4j. This task starts a timer at step 1002 to set the timer active variable at step 1004. A test is made at step 1006 to determine if the timer has expired. If so, the processing falls through to block 1008 where the timer variable is set inactive and then the task suspends itself at step 1010.

A Record Task 1100 is depicted in Figure 4k. This task is called whenever a memo is to be recorded. The task starts off by adding itself to the caller's conference at step 1102. Recording begins at step 1104 and continues until the task is halted, as tested at step 1106. When halted, the task stops recording and posts the message to the subscriber's voice mail (at step 1108) and then suspends (at step 1110).

Comparing the functionality of Figures 4a - 4k with the system described in Figure 2, those skilled in the art will, of course, appreciate that the system implementing the flow chart of Figures 4a - 4k is much more robust. In Figure 2, provision is made for the ability to park the caller and for a parked timer to expire. See, for example, steps 38 - 44. In the present multitasking embodiment described with reference to Figures 4a - 4k, there is no need to explicitly park the caller. If the person answering the telephone on the forwarding leg hangs up, that brings down the forwarding leg, but does not affect the caller on the calling leg 3. During that time, the Inbound Task would still be looping during the loop comprising the steps 220 - 225, or the loop comprising steps 217M - 225M.

Additional Features

Additional features are envisioned for the disclosed system. The hardware environment may include the ability to receive, decode and transmit SS-7 messages, as previously described. The SS-7 facilities being implemented in the PSTN can provide messages to the system 40 indicating, for example, that a subscriber

has turned on their cellular phone 81. Each time a cellular phone is turned on it must sign onto the cellular system 80 with which it communicates. The cellular system 80 can send a SS-7 message to system 40 informing the system 40 that the cellular telephone 81 has just signed on to the cellular system 80. The database of system 40 is used to determine how many Pager Tasks 300 need to be started in response to a caller's call (in addition to determining how those pages are to be made), and how many Forward Leg Tasks 500 need to be started in response to a callers call (in addition to determining to which telephone addresses the forward leg calls are to be addressed). This information may be stored as a normal schedule for the subscriber, as previously mentioned. The SS-7 messages, such as the message noted above, can be used to override the preprogrammed schedule. For example, at 10:00 A.M. weekdays the preprogrammed schedule "thinks" that the subscriber is in his, or her, office. However, at 10:05 A.M. the system receives an SS-7 message indicating that the cellular telephone in the subscriber's automobile was turned on. The database should indicate to the system how to react to this situation. Possible reactions might be:

(1) Do nothing;

(2) Switch Forwarding Tasks to the cellular telephone number, unless the subscriber is on the system;

(3) If the subscriber is on the system, emulate a Call Waiting function (preferably a priority call waiting function) and advise the subscriber in a new conference facility that their automobile (assuming that is where the cellular telephone in question is located) may be leaving the premises without their permission;

(4) During preprogrammed time periods, telephone the subscriber at one or more preprogrammed telephone numbers using the Forward Leg Tasks to tell them (using the digitized speech capabilities previously described) in a conference facility that their automobile (assuming that is where the cellular telephone in question is located) may be leaving the premises without their permission.

The SS-7 messaging service can also be used to advise the system 40 when the subscriber powers down the cellular telephone. The SS-7 signally specification does not explicitly provide a power down message, but the system 40 can learn about a power down situation by periodically sending a packet polling the cellular system about the status of the cellular telephone. If the cellular phone is no longer available, the system 40 will know that a power down must have occurred. The database preferably tells the system how to react. For example, then the system may revert back to the subscriber's preprogrammed schedule.

Those skilled in the art will appreciate that in the described system 40, when the subscriber is barging into a call, that they are recognized by the system as the subscriber and are given certain privileges. For example, in the disclosed embodiment, only the subscriber can force the forward leg(s) to disconnect by entering a code (#8) - see steps 266 and 267 on Figure 4b(5). It is felt that consumers resist having to always use PINs and therefore the disclosed system does not require a PIN on the forwarding leg - indeed it is felt that it would often be dysfunctional. However, one modification which probably would be accepted in the

marketplace would be to include the ability for the subscriber to enter an optional PIN on a forwarding leg. Thus, after the party answering the telephone on a forwarding leg enters the valid PIN for the subscriber who is being sought, they would be given subscriber status (i.e. the ability to force other forwarding legs to disconnect). Indeed, the monitoring mode could be disabled on the forward leg until the PIN were entered.

Another modification which might be desirable would be to automatically give subscriber status to persons answering the telephone at certain forwarding numbers (and perhaps only at certain times), as stored in the database. For example, if a call is forwarded to the telephone 76 on the subscriber's desk at their office, the database could tell the system to assume that the party answering that telephone is the subscriber and to give them subscriber status. On the other hand, if a call is forwarded to the telephone 16 at the subscriber's home, the database could tell the system to assume that the party answering that telephone is not the subscriber unless they enter a PIN. These assumptions could be, of course, reversed, by making appropriate entries in the database. Similarly, subscriber status could be automatically granted if the forward leg call is answered within a certain time limit or within a certain number of rings. Thus, it would be useful where the forward leg call goes to a subscriber's office and the subscriber has a no-answer forwarding featured turned on to forward the call to their secretary, say after two unanswered rings. If the subscriber is not in their office and the call is forwarded to their secretary after two rings, the subscriber status could be disabled by the third ring

(at the secretary's desk).

The changes needed to the disclosed flow charts to implement such capabilities are not extensive. For example, the forward leg on Figure 4e(3) could be enlarged to include a test for a PIN similar to the testing done at steps 248, 252 and 249 on Figure 4b(4). Thus, a step similar to step 252 would a test for a valid subscriber PIN, and to set a "subscriber on forwarding leg" variable true if detected (of course, the database could set that variable true without the need of the answeree to enter a PIN). The loop on Figure 4e(3) would then test additionally for the "subscriber on forwarding leg" variable at steps 562.1 and 570.

The processing on Figures 2b(5) and 5e(3) allow the subscriber to toggle between two conferences or to join two conferences together. Those skilled in the art realize that it is relatively straight forward to allow additional conferences to be set up. Indeed, since the outbound calling feature implemented by subroutines 277 and 577 require an additional conference to make an outbound call, it would be desirable to allow more than two conferences to be set up. The problem which arises is that humans are not particularly adept at keeping track of more than two conferences. Thus, while the toggling feature can easily be set up conceptually to stack the conferences, or arrange them in a ring, and to merely proceed to the next conference in the stack, or the ring, when the subscriber enters the ** keys, the problem is that a human is apt to forget who is in which conference. There is a possible solution to this problem. The system should be able to identify who, and where, and keep track of that information for the

subscriber. For example, using CLID and a database, the system might know the name (or some other identifying feature) of a caller. At least the caller's telephone number should be known. Similarly with outbound calls -- since the system preferably looks up the destination phone number in a database (see step 277.2 or 577.2), then the system also has identifying information available to it about outbound calls as well. This identifying information can be supplied to the subscriber as the caller switches (toggles) among active conferences. When switching conferences, the system plays a joining tone in the switched-to conference at steps 265.1 and 565.1. At more or less the same time the system could also generate sounds, or speech, telling the subscriber which conference is being entered. For example, the system might play "Home", "Your Secretary", or "Ms. Anderson" as the subscriber switches. The messages might include a conference number, thusly: "One, Home", "Two, Your Secretary", and, "Three, Ms. Anderson". The conference joining feature (steps 270, 271, 272 and 272.1 or 570, 570.1, 571, 572 and 572.1) may be modified to play the identifying information for each active conference, and ask the subscriber which conferences should be joined together. For example, if three conferences were active, upon entering the #3 keys the system might respond as follows:

(1) By saying: "There are three active conferences which may be joined together. Should "Home" (or "One Home") be joined?"

(2) The system would then wait for the subscriber to enter a * key (meaning yes), or a # key (meaning no). The testing would be done in a loop similar to steps 243 and 244.

(3) Next the system might generate: "Thank you.

Should "Your Secretary" (or "Two Your Secretary") be joined?

(4) Step (2) above is then repeated for conference number two.

(5) Step (3) is then repeated for conference number three.

(6) Step (2) is again repeated for the last conference.

(7) The conferences for which the subscriber entered the "yes" key (i.e., the * key in this example), would then be joined with a joining tone being played as at steps 272.1 and 572.1.

Having described a preferred embodiment of the invention, further modification and changes may now suggest themselves to those skilled in the art. The invention, therefore, is not to be limited to the disclosed embodiments as many changes and modifications will doubtless occur. Rather, the invention is to be defined by the scope of the accompanying claims.

APPENDIX I

SOURCE CODE LISTING OF MAIN PROGRAM WRITTEN IN VOS

```
#    MAIN - main program
#
#    This main task will spawn tasks for 12 time slots.
#    As calls enter the system, they will be routed to
an available voice
#    processing time slot via the SS96/SB96. Up to four
inbound calls to a
#    caller access number and a barge in call can be
handled concurrently.
#
#    GETDID+BARGER+CALLIN handle inbound calls
#    GETDID+CALLOUT handle barger initiated outbound
calls
#    FORWARD handles outbound forwarding calls
#    PAGE handles outbound paging calls
#    VMAIL handles outbound voice mail calls
#
#    The assumed configuration is an SS96/SB96 cabled to
a D121 on PEB1,
#    and cabled to DTI101 network cards on PEB2 and
PEB4.
#
#-----------------------------------------------------
-----------------------

dec
    include "datmod.inc"   # bring in library
definitions
    include "global.inc"   # bring in our global
definitions

    var code:16 ;      # return value from C library
calls
    var test:10, key:3 ;   # diagnostic commands
    var vpline:3 ;         # voice processor line number
    var netline:3 ;        # network line number
    var subs:3 ;       # subscriber number
    var call:3 ;       # call number
    var conf:3 ;       # conference number
    var vppids[GDIDBGN..PAGEEND]:3 ;  # pid for each
task controlling a D121 line
    var vplines[GDIDBGN..PAGEEND]:3 ;  # holds netline
number D121 line connected to
    var netlines[NETBGN..NETEND]:3 ;  # holds vpline
```

```
number network line connected to
    var altlines[ALTBGN..ALTEND]:3 ;  # holds vpline
number alternate netline connected to
    var confers[1..CONFMAX]:3 ;     # conference active
flags

    var line:3 ;       # needed for scr_stat
end

program

# Initializations

    # make sure we have enough global memory allocated
in INIT.DEF
    glb_set(GLOBALS-1, "123456789") ;
    if (glb_get(GLOBALS-1) strneq "123456789")
        voslog("SERIOUS ERROR: not enough global memory
allocated") ;
    endif

    # initialize active global variables to null
    for (code = 0; code < GLOBALS; code++)
        glb_set(code, "") ;
    endfor

    glb_set(STRATUS_PRE, "M*81234567") ;     # MF - PEB4
    glb_set(STRATUS_PST, "#") ;          # MF - PEB4

    subs = 0 ;                  # subscriber #1 (Bob
Fuller)

    glb_set(PSTN_PRE+subs, "T") ;        # DTMF - PEB2
    glb_set(PSTN_PST+subs, "") ;         # DTMF - PEB2
    glb_set(CALLER_NUMB+subs, "T206*654*5783#") ;
    glb_set(PAGER+subs, "6804910") ;     # Bob Fuller's
pager
    glb_set(PRGRM_PIN+subs, "123") ;
    glb_set(BARGE_PIN+subs, "*") ;
    glb_set(CALLER_DID+subs, "5832000") ;    # DTMF
    glb_set(BARGE_DID+subs, "6545783") ;     # DTMF
    glb_set(FORWARD+subs, "5832000") ;  # DTMF
    glb_set(VOICEMAIL+subs, "5832000") ;     # DTMF

    subs = 1 ;                  # subscriber #2 (Ron
Brooks)

    glb_set(PSTN_PRE+subs, "T") ;        # DTMF - PEB2
    glb_set(PSTN_PST+subs, "") ;         # DTMF - PEB2
    glb_set(CALLER_NUMB+subs, "T206*654*5781#") ;
    glb_set(PAGER+subs, "9171284") ;     # Ron Brooks'
```

```
pager
     glb_set(PRGRM_PIN+subs, "123") ;
     glb_set(BARGE_PIN+subs, "*") ;
     glb_set(CALLER_DID+subs, "6545780") ;     # DTMF
     glb_set(BARGE_DID+subs, "6545781") ;      # DTMF
     glb_set(FORWARD+subs, "6545780") ;  # DTMF
     glb_set(VOICEMAIL+subs, "6545780") ;      # DTMF

     subs = 2 ;                    # subscriber #3 (Jim
Brown)

     glb_set(PSTN_PRE+subs, "T") ;         # DTMF - PEB2
     glb_set(PSTN_PST+subs, "") ;          # DTMF - PEB2
     glb_set(CALLER_NUMB+subs, "T206*654*5785#") ;
     glb_set(PAGER+subs, "9916101") ;      # Jim Brown's
fast pager
     glb_set(PRGRM_PIN+subs, "123") ;
     glb_set(BARGE_PIN+subs, "*") ;
     glb_set(CALLER_DID+subs, "6541022") ;     # DTMF
     glb_set(BARGE_DID+subs, "6545785") ;      # DTMF
     glb_set(FORWARD+subs, "6541022") ;  # DTMF
     glb_set(VOICEMAIL+subs, "6541022") ;      # DTMF

     subs = 3 ;                    # subscriber #4 (Dan
Kransler)

     glb_set(PSTN_PRE+subs, "T") ;         # DTMF - PEB2
     glb_set(PSTN_PST+subs, "") ;          # DTMF - PEB2
     glb_set(CALLER_NUMB+subs, "T206*654*1003#") ;
     glb_set(PAGER+subs, "9972575") ;      # Dan
Kransler's fast pager
     glb_set(PRGRM_PIN+subs, "123") ;
     glb_set(BARGE_PIN+subs, "*") ;
     glb_set(CALLER_DID+subs, "6541000") ;     # DTMF
     glb_set(BARGE_DID+subs, "6541003") ;      # DTMF
     glb_set(FORWARD+subs, "6541000") ;  # DTMF
     glb_set(VOICEMAIL+subs, "6541000") ;      # DTMF

     subs = 4 ;                    # subscriber #5 (Ron
Brooks)

     glb_set(PSTN_PRE+subs, "T81234567") ;     # DTMF -
PEB2
     glb_set(PSTN_PST+subs, "") ;          # DTMF - PEB2
     glb_set(CALLER_NUMB+subs, "T206*654*5211#") ;
     glb_set(PAGER+subs, "9171284") ;      # Ron Brooks'
pager
     glb_set(PRGRM_PIN+subs, "123") ;
     glb_set(BARGE_PIN+subs, "*") ;
     glb_set(CALLER_DID+subs, "6545210") ;     # DTMF
     glb_set(BARGE_DID+subs, "6545211") ;      # DTMF
```

```
    glb_set(FORWARD+subs, "6545780") ;  # DTMF
    glb_set(VOICEMAIL+subs, "6545780") ;     # DTMF

    subs = 5 ;                  # subscriber #6 (Test
call)

    glb_set(PSTN_PRE+subs, "T") ;        # DTMF - PEB2
    glb_set(PSTN_PST+subs, "") ;         # DTMF - PEB2
    glb_set(CALLER_NUMB+subs, "T206*654*5798#") ;
    glb_set(PAGER+subs, "") ;       # no pager
    glb_set(PRGRM_PIN+subs, "123") ;
    glb_set(BARGE_PIN+subs, "*") ;
    glb_set(CALLER_DID+subs, "6545799") ;    # DTMF
    glb_set(BARGE_DID+subs, "6545798") ;     # DTMF
    glb_set(FORWARD+subs, "4543864") ;  # DTMF
    glb_set(VOICEMAIL+subs, "3927698") ;     # DTMF

    subs = 6 ;                  # subscriber #7 (Ron
Brooks)

    glb_set(PSTN_PRE+subs, "T") ;        # DTMF - PEB2
    glb_set(PSTN_PST+subs, "") ;         # DTMF - PEB2
    glb_set(CALLER_NUMB+subs, "T206*654*5787#") ;
    glb_set(PAGER+subs, "9171284") ;     # Ron Brooks'
pager
    glb_set(PRGRM_PIN+subs, "123") ;
    glb_set(BARGE_PIN+subs, "*") ;
    glb_set(CALLER_DID+subs, "6545786") ;    # DTMF
    glb_set(BARGE_DID+subs, "6545787") ;     # DTMF
    glb_set(FORWARD+subs, "3927698") ;  # DTMF
    glb_set(VOICEMAIL+subs, "6545780") ;     # DTMF

    for (subs = 0; subs < SUBSMAX; subs++)
        # try open programming PIN file
        code = fil_open("program"&subs&".nmb", "rs") ;

        # check if file opened
        if (code >= 0)
            # seek start of file
            fil_seek(code, 0, 0) ;

            # set programming PIN from file
            glb_set(PRGRM_PIN+subs, fil_getline(code))
;

            # close file
            fil_close(code) ;
        endif

        # try open barge in PIN file
        code = fil_open("bargein"&subs&".nmb", "rs") ;
```

```
          # check if file opened
          if (code >= 0)
               # seek start of file
               fil_seek(code, 0, 0) ;

               # set barge in PIN from file
               glb_set(BARGE_PIN+subs, fil_getline(code))
;

               # close file
               fil_close(code) ;
          endif

          # try open forwarding number file
          code = fil_open("forward"&subs&".nmb", "rs") ;

          # check if file opened
          if (code >= 0)
               # seek start of file
               fil_seek(code, 0, 0) ;

               # set forwarding number from file
               glb_set(FORWARD+subs, fil_getline(code)) ;

               # close file
               fil_close(code) ;
          endif

          # try open voice mail number file
          code = fil_open("vmail"&subs&".nmb", "rs") ;

          # check if file opened
          if (code >= 0)
               # seek start of file
               fil_seek(code, 0, 0) ;

               # set voice mail number from file
               glb_set(VOICEMAIL+subs, fil_getline(code))
;

               # close file
               fil_close(code) ;
          endif
     endfor

#-----------------------------------------------------
----------------------

     # reset the SS96 to load parameters from datmod.par
file
```

```
    voslog("Sreset() returns "&Sreset()) ;

    # display the SS96 conference counts
    voslog("Conferences: "&Sgetmode(8)) ;

    # mark all vplines as free
    for (vpline = GDIDBGN ; vpline <= PAGEEND ;
vpline++)
        if ((code = Ssetsig(vpline, SSAS_ONHOOK,
SSAL_NO)) <> 0)
            voslog("Ssetsig("&vpline&") code "&code&"
error") ;
        endif
        vplines[vpline] = 0 ;
    endfor

    # make our pid public
    glb_set(MAINPID, getpid()) ;
    # initialize go-ahead flag
    glb_set(MAINFLG, getpid()) ;

    # spawn line tasks to control D121 vplines
    for (vpline = GDIDBGN ; vpline <= PAGEEND ;
vpline++)
        switch (vpline)

        case GDIDBGN:     spawna("getdid", vpline) ;
        case GDIDBGN+1:       spawna("getdid", vpline)
;
        case GDIDBGN+2:       spawna("getdid", vpline)
;
        case GDIDEND:     spawna("getdid", vpline) ;

        case FWRDBGN:     spawna("forward", vpline) ;
        case FWRDBGN+1:       spawna("forward",
vpline) ;
        case FWRDBGN+2:       spawna("forward",
vpline) ;
        case FWRDEND:     spawna("forward", vpline) ;

        case MAILBGN:     spawna("vmail", vpline) ;
        case MAILEND:     spawna("vmail", vpline) ;

        case PAGEBGN:     spawna("page", vpline) ;
        case PAGEEND:     spawna("page", vpline) ;
    default:
        endswitch

        # get PID of task for this line
        vppids[vpline] = msg_get(2) ;
    endfor
```

```
     # change flag to "go ahead" value
     glb_set(MAINFLG, 99) ;

     # initialize signal alerting on network channels
     for (netline = NETBGN ; netline <= NETEND ;
netline++)
          # set onhook with signal alerting enabled
          if ((code = Ssetsig(netline, SSAS_ONHOOK,
SSAL_YES)) <> 0)
               voslog("Ssetsig("&netline&") code "&code&"
error") ;
          endif
          # set conferencing attenuation and suppression
          if ((code = Scnfparm(netline, SBAT_NONE,
SBSU_SLIGHT)) <> 0)
               voslog("Scnfparm("&netline&") code
"&code&" error") ;
          endif
          netlines[netline] = 0 ;
     endfor

     # initialize signal alerting on alternate network
channels
     for (netline = ALTBGN ; netline <= ALTEND ;
netline++)
          # set onhook with signal alerting disabled
          if ((code = Ssetsig(netline, SSAS_ONHOOK,
SSAL_NO)) <> 0)
               voslog("Ssetsig("&netline&") code "&code&"
error") ;
          endif
          # set conferencing attenuation and suppression
          if ((code = Scnfparm(netline, SBAT_NONE,
SBSU_SLIGHT)) <> 0)
               voslog("Scnfparm("&netline&") code
"&code&" error") ;
          endif
          altlines[netline] = 0 ;
     endfor

     # draw initial debug screen display
     scr_init() ;

# Main loop
     for(;;)
          # check event queue on SS96
          code = Sgetevt(0) ;
          switch( substr(code, 1, 1))     # process based
on first character
```

```
        case "E":     # valid event
             netline = substr(code, 3, 2) ;  # get line
event occured on
             switch(substr(code, 5, 2)) # process based
on type of event

             case SSEVT_SIGNAL:     # signal bit changed
                  # event on network line ?
                  if ((netline >= NETBGN) and (netline
<= NETEND))
                       voslog("SS Event: "&code) ;
                       # check received bit state
                       if (substr(code,7,1) eq 1)
                            # check for existing call on
netline
                            if (netlines[netline] <> 0)
                                 # existing call - assume
this is answer supervision
                                 # check for forwarding
task vpline
                                 if ((netlines[netline]
>= FWRDBGN) and (netlines[netline] <= FWRDEND))
                                      # look through
subscribers and calls for call progress active on
netline
                                      for (subs = 0; subs
< SUBSMAX; subs++)
                                           for (call = 0;
call < FWRDMAX; call++)
                                                if
(glb_get(FWRD1CPA+SUBSMAX*call+subs) eq netline)
                                                     #
abort possible caller's task sc_play (e.g. ringing)
sc_abort(glb_get(CALL1ACT+SUBSMAX*call+subs)) ;
voslog("sc_abort("&glb_get(CALL1ACT+SUBSMAX*call+su
bs)&")") ;
                                                     #
abort forwarding call progress analysis
sc_abort(netlines[netline]) ;
voslog("sc_abort("&netlines[netline]&")") ;
                                                     #
request answer supervision be returned to caller
glb_set(CALL1ANS+SUBSMAX*call+subs, vpline) ;
                                                endif
```

```
                                        endfor
                                    endfor
                                endif
                            else
                                # new incoming call if
signaling changed to offhook (ring)

                                # Find a free inbound
vpline, if available
                                for (vpline = GDIDBGN;
(vpline <= GDIDEND) and (vplines[vpline] <> 0);
vpline++)
                                endfor
                                if (vpline > GDIDEND)
                                    voslog("ERROR: no
free inbound vpline") ;
                                else
                                    # free vpline found

                                    # mark lines as busy
                                    vplines[vpline] =
netline ;
                                    netlines[netline] =
vpline ;

                                    # inform GETDID task
what vpline to use
                                    if ((code =
msg_put(vppids[vpline], vpline)) <> 0)
                                        voslog
("msg_put("&vppids[vpline]&","&vpline&") code "&code&"
error") ;
                                    endif

                                    # inform GETDID task
what netline to process
                                    if ((code =
msg_put(vppids[vpline], netline)) <> 0)
                                        voslog
("msg_put("&vppids[vpline]&","&netline&") code "&code&"
error") ;
                                    endif

                                    # inform GETDID task
the call direction to process
                                    if ((code =
msg_put(vppids[vpline], "In")) <> 0)
                                        voslog
("msg_put("&vppids[vpline]&",In) code "&code&" error") ;
                                    endif
                                endif
```

```
                    endif
                endif
            else
                # event on alternate network line
?
                if ((netline >= ALTBGN) and
(netline <= ALTEND))
                    voslog("SS Event: "&code) ;
                    # check received bit state
                    if (substr(code,7,1) eq 1)
                        # check for existing
call on alternate netline
                        if (altlines[netline] <>
0)
                            # existing call -
assume this is answer supervision
                            # check for
forwarding task vpline
                            if
((altlines[netline] >= FWRDBGN) and (altlines[netline]
<= FWRDEND))
                                # look through
subscribers and call for call progress active on netline
                                for (subs = 0;
subs < SUBSMAX; subs++)
                                    for (call =
0; call < FWRDMAX; call++)
                                        if
(glb_get(FWRD1CPA+SUBSMAX*call+subs) eq netline)
                                            #
abort possible caller's task sc_play (e.g. ringing)
sc_abort(glb_get(CALL1ACT+SUBSMAX*call+subs)) ;
voslog("sc_abort("&glb_get(CALL1ACT+SUBSMAX*call+su
bs)&")") ;

                                            #
abort forwarding call progress analysis
sc_abort(altlines[netline]) ;
voslog("sc_abort("&altlines[netline]&")") ;

                                            #
request answer supervision be returned to caller
glb_set(CALL1ANS+SUBSMAX*call+subs, vpline) ;
                                        endif
                                    endfor
                                endfor
```

```
                                        endif
                                    endif
                                endif
                            else
                                voslog("ERROR: "&code&" -
event not on network line") ;
                            endif
                        endif

                case SSEVT_SBOVERFLOW: # conference
overflow
                    voslog("SS Event: "&code&" -
conference overflow") ;

                case SSEVT_FLIPDONE:   # flip command
completed
                    voslog("SS Event: "&code&" - flip
command completed") ;

                case SSEVT_RECVWINK:   # inbound wink
detected
                    voslog("SS Event: "&code&" - inbound
wink detected") ;

                case SSEVT_RECVFLASH:  # inbound flash
detected
                    voslog("SS Event: "&code&" - inbound
flash detected") ;

                case SSEVT_TIMEOUT:    # line's timer
expired
                    voslog("SS Event: "&code&" - line's
timer expired") ;

                default:       # unexpected event
                    voslog("ERROR: "&code&" - unexpected
event") ;

                endswitch

            case "N":      # no event found
                if (kb_qsize() > 0)
                    key = kb_get() ;
                    switch (key)
                    case "s":      # SmartSwitch only
                        voslog("ERROR: SS
"&substr(1000+test,2,3)&":"&Sstatus(test)) ;
                        test = 0 ;
                    default:
                        if ((key >= 0) and (key <= 9))
                            test = 10*test + key ;
```

```
                    else
                        test = 0 ;
                    end
                endswitch
            end

        case "X": # error
        default: # error - should never happen
            voslog("ERROR: Sgetevt() returned error =
"&code) ;

        endswitch

    # Check for messages from line task
        code = msg_get(0) ;    # don't wait if message
not pending
        switch (substr(code, 1, 1))

        case "O": # allocate and start an outbound
CALLER task
            call = substr(code, 2, 1) ;     # get
message call number
            netline = substr(code, 3, 2) ; # get
message netline number
            subs = substr(code, 5, 99) ;    # get
message subscriber number
            # Find a free inbound vpline, if available
            for (vpline = GDIDBGN; (vpline <= GDIDEND)
and (vplines[vpline] <> 0); vpline++)
            endfor
            if (vpline > GDIDEND)
                voslog("ERROR: no free outbound
vpline") ;
                if ((code = msg_put(msg_pid(), "O-
NO")) <> 0)
                    voslog("msg_put("&msg_pid()&",O-
NO) code "&code&" error") ;
                endif
            else
                # free vpline found
                # mark lines as busy
                vplines[vpline] = netline ;
                if ((netline >= NETBGN) and (netline
<= NETEND))
                    netlines[netline] = vpline ;
                else
                    if ((netline >= ALTBGN) and
(netline <= ALTEND))
                        altlines[netline] = vpline ;
                    else
                        voslog("ERROR: invalid
```

```
outbound netline "&netline) ;
                    endif
                endif
                # inform GETDID task what vpline to
use
                if ((code = msg_put(vppids[vpline],
vpline)) <> 0)
                    voslog("msg_put("&vppids[vpl
ine]&","&vpline&") code "&code&" error") ;
                endif

                # inform GETDID task what netline to
use
                if ((code = msg_put(vppids[vpline],
netline)) <> 0)
                    voslog("msg_put("&vppids[vpl
ine]&","&netline&") code "&code&" error") ;
                endif

                # inform GETDID task what direction to
use
                if ((code = msg_put(vppids[vpline],
"Out")) <> 0)
                    voslog("msg_put("&vppids[vpl
ine]&",Out) code "&code&" error") ;
                endif

                # inform GETDID task what subscriber
to use (when dir is "Out")
                if ((code = msg_put(vppids[vpline],
subs)) <> 0)
                    voslog("msg_put("&vppids[vpl
ine]&","&subs&") code "&code&" error") ;
                endif

                # inform GETDID task what call to use
(when dir is "Out")
                if ((code = msg_put(vppids[vpline],
call)) <> 0)
                    voslog("msg_put("&vppids[vpl
ine]&","&call&") code "&code&" error") ;
                endif

                # reply to requesting task
                if ((code = msg_put(msg_pid(),
"O"&vpline)) <> 0)
                    voslog("msg_put("&msg_pid()&",O
"&vpline&") code "&code&" error") ;
                endif
            endif
```

```
        case "F": # allocate and start a "forwarding"
vpline task
                call = substr(code, 2, 1) ;     # get
message call number
                subs = substr(code, 3, 99) ;    # get
message subscriber number
                if ((call < 0) or (call >= CALLMAX))
                     voslog("ERROR: invalid call number
"&call&" message received") ;
                else
                     if ((subs < 0) or (subs >= SUBSMAX))
                          voslog("ERROR: invalid subscriber
number "&subs&" message received") ;
                     else
                          # Find a free "forwarding"
vpline, if available
                          for (vpline = FWRDBGN; (vpline <=
FWRDEND) and (vplines[vpline] <> 0); vpline++)
                          endfor
                          if (vpline > FWRDEND)
                               voslog("ERROR: no free
forwarding vpline") ;
                          else
                               # free vpline found - mark it
as busy
                               vplines[vpline] = vpline ;

                               # inform requesting task what
vpline was assigned
                               if ((code =
msg_put(msg_pid(), "F"&vpline)) <> 0)
                                    voslog("msg_put
("&msg_pid()&",F"&vpline&") code "&code&" error") ;
                               endif

                               # inform FORWARD task what
vpline to use
                               if ((code =
msg_put(vppids[vpline], vpline)) <> 0)
                                    voslog("msg_put
("&vppids[vpline]&","&vpline&") code "&code&" error") ;
                               endif

                               # inform FORWARD task of
related subscriber number
                               if ((code =
msg_put(vppids[vpline], subs)) <> 0)
                                    voslog("msg_put
("&vppids[vpline]&","&subs&") code "&code&" error") ;
                               endif
```

```
                        # inform FORWARD task of
related call number
                        if ((code =
msg_put(vppids[vpline], call)) <> 0)
                            voslog("msg_put
("&vppids[vpline]&","&call&") code "&code&" error") ;
                        endif
                    endif
                endif
            endif

        case "V": # allocate and start a "voice mail"
vpline task
            call = substr(code, 2, 1) ;     # get
message call number
            subs = substr(code, 3, 99) ;    # get
message subscriber number
            if ((call < 0) or (call >= CALLMAX))
                voslog("ERROR: invalid call number
"&call&" message received") ;
            else
                if ((subs < 0) or (subs >= SUBSMAX))
                    voslog("ERROR: invalid subscriber
number "&subs&" message received") ;
                else
                    # Find a free "voice mail"
vpline, if available
                    for (vpline = MAILBGN; (vpline <=
MAILEND) and (vplines[vpline] <> 0); vpline++)
                    endfor
                    if (vpline > MAILEND)
                        voslog("ERROR: no free voice
mail vpline") ;
                    else
                        # free vpline found - mark it
as busy
                        vplines[vpline] = vpline ;

                        # inform requesting task what
vpline was assigned
                        if ((code =
msg_put(msg_pid(), "V"&vpline)) <> 0)
                            voslog("msg_put
("&msg_pid()&",V"&vpline&") code "&code&" error") ;
                        endif

                        # inform VMAIL task what
vpline to use
                        if ((code =
msg_put(vppids[vpline], vpline)) <> 0)
                            voslog("msg_put
```

```
("&vppids[vpline]&","&vpline&") code "&code&" error") ;
                              endif

                              # inform VMAIL task of
related subscriber number
                              if ((code =
msg_put(vppids[vpline], subs)) <> 0)
                                   voslog("msg_put
("&vppids[vpline]&","&subs&") code "&code&" error") ;
                              endif

                              # inform VMAIL task of
related call number
                              if ((code =
msg_put(vppids[vpline], call)) <> 0)
                                   voslog("msg_put
("&vppids[vpline]&","&call&") code "&code&" error") ;
                              endif
                         endif
                    endif
               endif

          case "P": # allocate and start a "paging"
vpline task
               subs = substr(code, 2, 99) ;    # get
message subscriber number
               if ((subs < 0) or (subs >= SUBSMAX))
                    voslog("ERROR: invalid subscriber
number "&subs&" message received") ;
               else
                    # Find a free "paging" vpline, if
available
                    for (vpline = PAGEBGN; (vpline <=
PAGEEND) and (vplines[vpline] <> 0); vpline++)
                    endfor
                    if (vpline > PAGEEND)
                         voslog("ERROR: no free paging
vpline") ;
                    else
                         # free vpline found - mark it as
busy
                         vplines[vpline] = vpline ;

                         # inform requesting task what
vpline was assigned
                         if ((code = msg_put(msg_pid(),
"P"&vpline)) <> 0)
                              voslog("msg_put("&msg_pid
()&",P"&vpline&") code "&code&" error") ;
                         endif
```

```
                    # inform PAGE task what vpline to
use
                    if ((code =
msg_put(vppids[vpline], vpline)) <> 0)
                        voslog("msg_put("&vppids
[vpline]&","&vpline&") code "&code&" error") ;
                    endif

                    # inform PAGE task of related
subscriber number
                    if ((code =
msg_put(vppids[vpline], subs)) <> 0)
                        voslog("msg_put("&vppids
[vpline]&","&subs&") code "&code&" error") ;
                    endif
                endif
            endif

        case "N": # allocate netline
            vpline = substr(code, 2, 99) ; # get
message vpline

            # Find a free netline, if available -
search from back to front
            for (netline = NETEND; (netline >= NETBGN)
and (netlines[netline] <> 0); netline--)
            endfor
            if (netline < NETBGN)
                voslog("ERROR: no free netline to
allocate") ;
                if ((code = msg_put(msg_pid(), "N-
NO")) <> 0)
                    voslog("msg_put("&msg_pid()&",N-
NO) code "&code&" error") ;
                endif
            else
                netlines[netline] = vpline ;    # mark
network line busy
                vplines[vpline] = netline ;     #
update vpline connection data

                # reply to requesting task
                if ((code = msg_put(msg_pid(), "N-
OK")) <> 0)
                    voslog("msg_put("&msg_pid()&",N-
OK) code "&code&" error") ;
                endif

                # send message to inform task of
netline allocated
                if ((code = msg_put(msg_pid(),
```

```
netline)) <> 0)
                    voslog("msg_put("&msg_pid()&",
"&netline&") code "&code&" error") ;
                endif
            endif

        case "A": # allocate alternate netline
            vpline = substr(code, 2, 99) ; # get
message vpline

            # Find a free netline, if available -
search from front to back
            for (netline = ALTBGN; (netline <= ALTEND)
and (altlines[netline] <> 0); netline++)
            endfor
            if (netline > ALTEND)
                voslog("ERROR: no free netline to
allocate") ;
                if ((code = msg_put(msg_pid(), "A-
NO")) <> 0)
                    voslog("msg_put("&msg_pid()&",A-
NO) code "&code&" error") ;
                endif
            else
                altlines[netline] = vpline ;    # mark
network line busy
                vplines[vpline] = netline ;     #
update vpline connection data

                # reply to requesting task
                if ((code = msg_put(msg_pid(), "A-
OK")) <> 0)
                    voslog("msg_put("&msg_pid()&",A-
OK) code "&code&" error") ;
                endif

                # send message to inform task of
netline allocated
                if ((code = msg_put(msg_pid(),
netline)) <> 0)
                    voslog("msg_put("&msg_pid()&",
"&netline&") code "&code&" error") ;
                endif
            endif

        case "D": # deallocate netline
            netline = substr(code, 2, 2) ; # get
message netline
            vpline = substr(code, 4, 99) ; # get
message vpline
```

```
                # check for vpline deallocate request
                if (vpline <> 0)
                     # mark vpline as free
                     vplines[vpline] = 0 ;
                endif

                if ((netline >= NETBGN) and (netline <=
NETEND))
                     if ((vpline <> 0) and
(netlines[netline] <> vpline))
                          voslog("ERROR: No active netline
"&netline&" to deallocate") ;
                          if ((code = msg_put(msg_pid(),
"D-NO")) <> 0)
                               voslog("msg_put("&msg_pid
()&",D-NO) code "&code&" error") ;
                          endif
                     else
                          # mark netline as free
                          netlines[netline] = 0 ;

                          # reply to requesting task
                          if ((code = msg_put(msg_pid(),
"D-OK")) <> 0)
                               voslog("msg_put("&msg_pid
()&",D-OK) code "&code&" error") ;
                          endif
                     endif
                else
                     if ((netline >= ALTBGN) and (netline
<= ALTEND))
                          if ((vpline <> 0) and
(altlines[netline] <> vpline))
                               voslog("ERROR: No active
alternate netline "&netline&" to deallocate") ;
                               if ((code =
msg_put(msg_pid(), "D-NO")) <> 0)
                                    voslog("msg_put
("&msg_pid()&",D-NO) code "&code&" error") ;
                               endif
                          else
                               # check for vpline deallocate
request
                               if (vpline <> 0)
                                    # mark vpline as free
                                    vplines[vpline] = 0 ;
                               endif

                               # mark alternate netline as
free
                               altlines[netline] = 0 ;
```

```
                                # reply to requesting task
                                if ((code =
msg_put(msg_pid(), "D-OK")) <> 0)
                                    voslog("msg_put
("&msg_pid()&",D-OK) code "&code&" error") ;
                                endif
                            endif
                        else
                            voslog("ERROR: in deallocating
netline "&netline) ;
                            continue ;
                        endif
                    endif

                case "C": # allocate conference
                    # Find a free conference, if available
                    for (conf = 1; (conf <= CONFMAX) and
(confers[conf] <> 0); conf++)
                    endfor
                    if (conf > CONFMAX)
                        voslog("ERROR: no free conference to
allocate") ;
                        if ((code = msg_put(msg_pid(), "C-
NO")) <> 0)
                            voslog("msg_put("&msg_pid()&",C-
NO) code "&code&" error") ;
                        endif
                    else
                        # set conference active
                        confers[conf] = conf ;

                        # reply to requesting task
                        if ((code = msg_put(msg_pid(), "C-
OK")) <> 0)
                            voslog("msg_put("&msg_pid()&",C-
OK) code "&code&" error") ;
                        endif

                        # send message to inform task of
conference allocated
                        if ((code = msg_put(msg_pid(), conf))
<> 0)
                            voslog("msg_put("&msg_pid()&",
"&conf&") code "&code&" error") ;
                        endif
                    endif

                case "E": # deallocate conference
                    conf = substr(code, 2, 99) ;    # get
message conference
```

```
                if ((conf < 1) or (conf > CONFMAX))
                    voslog("ERROR: in deallocating conf
"&conf) ;
                    if ((code = msg_put(msg_pid(), "E-
NO")) <> 0)
                        voslog("msg_put("&msg_pid()&",E-
NO) code "&code&" error") ;
                    endif
                else
                    # mark conference as free
                    confers[conf] = 0 ;

                    # reply to requesting task
                    if ((code = msg_put(msg_pid(), "E-
OK")) <> 0)
                        voslog("msg_put("&msg_pid()&",E-
OK) code "&code&" error") ;
                    endif
                endif
            endswitch
        endfor
end
```

APPENDIX II

SOURCE CODE LISTING OF AN INBOUND TASK
USED BY THE MAIN PROGRAM

```
#
#    BARGER.VS - D121 line task
#
#    Designed to work in concert with GETDID.VS
#
#-------------------------------------------------------
------------------------

dec
     include "datmod.inc"          # SS96 library
definitions
     include "global.inc"          # define global variable
numbers

     const NOCONF = 0 ;            # check_waiting() not in
conference
     const TALKER = 1 ;            # check_waiting()
conference talker
     const LISTENER = 2 ;          # check_waiting()
conference listener

     var code:16 ;               # return value from function
calls
     var vpline:3 ;                # voice processing line
number
     var netline:3 ;               # inbound network line
number
     var outline:3 ;               # outbound network line
number
     var subs:3 ;                # subscriber number
     var call:3 ;                # call number
     var flash:3 ;               # flash DTMF digits
     var mainpid:3 ;                # main task PID
     var sdate:7 ;               # call starting date
     var stime:7 ;               # call starting time
     var pin_entry:10 ;             # barger PIN entry
     var phone_num:16 ;             # phone number input
     var file:3 ;                # file handle

     var conf:3 ;                # conference number
 (confsubs)
```

```
     var stats:16 ;
status (confsubs)                    # beginning netline
     var n:6, line:3 ;          # needed for scr_stat
end

program

# Initializations
     mainpid = glb_get(MAINPID) ;

     vpline = arg() ;
     if ((vpline < GDIDBGN) or (vpline > GDIDEND))
          voslog("ERROR: invalid vpline number "&vpline&"
message received") ;
          hangup() ;
          chain("GETDID", vpline) ;
     endif
     line = vpline ;                              # debug only

# Begin call processing

     # do not allow disconnects to run onsignal
     sc_watch(vpline, "+-", 1) ;

     # enable hang up jump to onsignal
     sc_use(vpline) ;

     # set call number inactive (in case of unexpected
disconnect)
     call = "" ;

     while (length(netline = msg_get(3)) eq 0)
     endwhile
     if ((netline < NETBGN) or (netline > NETEND))
          voslog("ERROR: invalid netline number
"&netline&" message received") ;
          hangup() ;
          chain("GETDID", vpline) ;
     endif

     while (length(subs = msg_get(3)) eq 0)
     endwhile
     if ((subs < 0) or (subs >= SUBSMAX))
          voslog("ERROR: invalid subscriber number
"&subs&" message received") ;
          hangup() ;
          chain("GETDID", vpline) ;
     endif

     while (length(pin_entry = msg_get(3)) eq 0)
     endwhile
```

```
    # set task debug screen status
    scr_stat("B-"&netline) ;

    # set barger present with both calls inactive
    glb_set(BARG1SUB+subs, "0") ;
    glb_set(BARG2SUB+subs, "0") ;

    # return answer supervision
    while ((code = sc_offhook(vpline)) <> T_OFFH)
        voslog("ERROR: code "&code&" while taking
vpline "&vpline&" offhook") ;
        sleep(10) ;
    endwhile
    voslog("Answer supervision") ;

    # now allow disconnects to run onsignal
    sc_watch(vpline, "d+-", 1) ;

    # clear digit buffer
    sc_clrdigits(vpline) ;

    if ((pin_entry streq glb_get(BARGE_PIN+subs))
    or  (pin_entry streq glb_get(PRGRM_PIN+subs)))
        goto sel_loop ;
    endif

    # check for a caller waiting
    if ((glb_get(CALL1ACT+subs) <> 0)
     or (glb_get(CALL2ACT+subs) <> 0))
        # reset call waiting request
        glb_set(CALLWAIT+subs, "") ;

        # check for immediate barge with null barge PIN
        if (glb_get(BARGE_PIN+subs) streq "")
            goto barge_in ;
        else

            # play ringing message (allows tone
interruption)
            voslog("sc_play('ringing'(001).vox)") ;
            sc_play(vpline, "PR0001.vox") ;

            voslog("sc_play('call waiting'(105).vox)")
;
            sc_play(vpline, "PR0105.vox") ;
        endif
    else
        # play ringing message (allows tone
interruption)
        voslog("sc_play('ringing'(001).vox)") ;
```

```
        sc_play(vpline, "PR0001.vox") ;
    endif

    # start PIN entry loop elapsed time
    sdate = date() ;
    stime = time() ;

    # check for DTMF entry in first second
    if ((code = sc_getdigits(vpline, 1, 1, 1)) eq
T_MAXDT)
        # add digit to PIN entry string
        pin_entry = sc_digits(vpline) ;

        goto check_pin ;
    else
        # clear PIN entry strings
        pin_entry = "" ;

        goto case_3 ;
    endif

pin_loop:
    while (timesub(date(), time(), sdate, stime) <= 20)
        # check for call waiting request
        check_waiting(NOCONF) ;

        # clear PIN entry string
        pin_entry = "" ;

        # ask for PIN number entry
        voslog("sc_play('enter pin'(286).vox)") ;
        sc_play(vpline, "PR0286.vox") ;
get_pin:
        while ((code = sc_getdigits(vpline, 1, 3, 3))
<> T_TIME)
            # add digit to PIN entry string
            pin_entry = pin_entry&sc_digits(vpline) ;
check_pin:
            if ((pin_entry streq
glb_get(BARGE_PIN+subs))
            or  (pin_entry streq
glb_get(PRGRM_PIN+subs)))
                break ;
            endif
        endwhile

        if (code eq T_TIME)
            continue ;
        endif

        voslog("PIN entry: "&pin_entry) ;
```

```
sel_loop:
          # restart elapsed time
          sdate = date() ;
          stime = time() ;

          # check to see if entry matches programming
password
          if (pin_entry streq glb_get(PRGRM_PIN+subs))
               while (timesub(date(), time(), sdate,
stime) <= 20)
                    # clear digit buffer
                    sc_clrdigits(vpline) ;

                    # check for call waiting request
                    check_waiting(NOCONF) ;

                    # give programming selections
                    voslog("sc_play('selection'(D00
1).vox)") ;
                    sc_play(vpline, "D001.vox") ;

                    if ((code = sc_getdigits(vpline, 1, 2,
2)) eq T_MAXDT)
                         break;
                    endif
               endwhile
               # check for dropped out of entry loop due
to tone entry
               if (code <> T_MAXDT)
                    break ;
               endif

               # check for call waiting request
               check_waiting(NOCONF) ;

               code = sc_digits(vpline) ;
               voslog("Subscriber selection number:
"&code) ;

               switch (code)

               # For testing - become an inbound Caller
               case "#":
                    if (glb_get(CALLER_DID+subs) streq
"6545786")
                         # set barger not present with
both calls inactive
                         glb_set(BARG1SUB+subs, "") ;
                         glb_set(BARG2SUB+subs, "") ;

                         # send netline number to CALLIN
```

```
                    msg_put(getpid(), netline) ;
                    # send subscriber number to
CALLIN
                    msg_put(getpid(), subs) ;
                    # replace ourself with CALLIN
                    chain("CALLIN", vpline) ;
               endif

          # For testing - record message prompts
          case "6":
               if (glb_get(CALLER_DID+subs) streq
"6545786")
                    # play the prompt
                    sc_play(vpline, "record.vox") ;
                    # record a message
                    sc_record(vpline, "message.vox",
6, 1, 0) ;
                    sleep(5) ;
                    # play the message back
                    sc_play(vpline, "message.vox") ;
               endif

          # Change forwarding number
          case "1":
               # clear digit buffer
               sc_clrdigits(vpline) ;

               # ask for new forwarding number
               voslog("sc_play('enter transfer
number'(D005).vox)") ;
               sc_play(vpline, "D005.vox") ;

               # set '*' and '#' as mask digits
               sc_toneint(vpline, 1, "*#") ;

               if (((code = sc_getdigits(vpline, 16,
10, 3)) <> T_MAXDT) and (code <> T_MDTMF) and (code <>
T_SIL))
                    voslog("ERROR: "&code&" -
entering forwarding number failure") ;
               else
                    phone_num = sc_digits(vpline) ;

                    # allow entry of a second '#' or
'*'
                    if (((code = sc_getdigits(vpline,
1, 1, 1)) <> T_MAXDT) and (code <> T_MDTMF) and (code <>
T_SIL))
                         voslog("ERROR: "&code&" -
entering forwarding number failure") ;
                    endif
```

```
                    phone_num =
phone_num&sc_digits(vpline) ;
                    voslog("Subscriber forwarding
number: "&phone_num) ;

                    # check for leaving old number
                    if (phone_num strneq "**")
                         # remove any terminating '*'
or '#'
                         if ((substr(phone_num,
length(phone_num), 1) streq "*")
                              or (substr(phone_num,
length(phone_num), 1) streq "#"))
                                   phone_num =
substr(phone_num, 1, length(phone_num) - 1) ;
                         endif

                         # update subscriber's
forwarding number
                         glb_set(FORWARD+subs,
phone_num) ;

                         # open forwarding number file
                         file =
fil_open("forward"&subs&".nmb", "wcts") ;

                         # seek start of file
                         fil_seek(file, 0, 0) ;

                         # write new forwarding number
string to file
                         fil_writes(file,
glb_get(FORWARD+subs)&"`r`n") ;

                         # close file
                         fil_close(file) ;
                    endif

                    # indicate current transfer
number follows
                    voslog("sc_play('transfer number
is'(D004).vox)") ;
                    sc_play(vpline, "D004.vox") ;

                    # play current forwarding number
                    if (glb_get(FORWARD+subs) strneq
"")
                         voslog("spk_dgts('"&glb_get
(FORWARD+subs)&"')") ;
                         spk_dgts(glb_get(FORWARD
```

```
+subs)) ;
                        else
                            voslog("sc_play('null'
(????).vox)") ;
                            sc_play(vpline, "null.vox") ;
                        endif
                    endif

                    # enable tone interruption
                    sc_toneint(vpline, 1) ;

                # Change voice mail number
                case "2":
                    # clear digit buffer
                    sc_clrdigits(vpline) ;

                    # ask for new voice mail number
                    voslog("sc_play('enter voice mail
number'(D002).vox)") ;
                    sc_play(vpline, "D002.vox") ;

                    # set '*' and '#' as mask digits
                    sc_toneint(vpline, 1, "*#") ;

                    if (((code = sc_getdigits(vpline, 16,
10, 3)) <> T_MAXDT) and (code <> T_MDTMF) and (code <>
T_SIL))
                        voslog("ERROR: "&code&" -
entering voice mail number failure") ;
                    else
                        phone_num = sc_digits(vpline) ;

                        # allow entry of a second '#' or
'*'
                        if (((code = sc_getdigits(vpline,
1, 1, 1)) <> T_MAXDT) and (code <> T_MDTMF) and (code <>
T_SIL))
                            voslog("ERROR: "&code&" -
entering forwarding number failure") ;
                        endif

                        phone_num =
phone_num&sc_digits(vpline) ;
                        voslog("Subscriber voice mail
number: "&phone_num) ;

                        # check for leaving old number
                        if (phone_num strneq "**")
                            # remove any terminating '*'
or '#'
                            if ((substr(phone_num,
```

```
length(phone_num), 1) streq "*")
                                or (substr(phone_num,
length(phone_num), 1) streq "#"))
                                    phone_num =
substr(phone_num, 1, length(phone_num) - 1) ;
                               endif

                               # update subscriber's voice
mail number
                               glb_set(VOICEMAIL+subs,
phone_num) ;

                               # open voice mail number file
                               file =
fil_open("vmail"&subs&".nmb", "wcts") ;

                               # seek start of file
                               fil_seek(file, 0, 0) ;

                               # write new voice mail number
string to file
                               fil_writes(file,
glb_get(VOICEMAIL+subs)&"`r`n") ;

                               # close file
                               fil_close(file) ;
                          endif

                          # indicate current voice mail
number follows
                          voslog("sc_play('voice mail
number is'(D003).vox)") ;
                          sc_play(vpline, "D003.vox") ;

                          # play current voice mail number
                          if (glb_get(VOICEMAIL+subs)
strneq "")
                               voslog("spk_dgts('"&glb_get
(VOICEMAIL+subs)&"')") ;
                               spk_dgts(glb_get(VOICEMA
IL+subs)) ;
                          else
                               voslog("sc_play('null'
(????).vox)") ;
                               sc_play(vpline, "null.vox") ;
                          endif
                     endif

                     # enable tone interruption
                     sc_toneint(vpline, 1) ;
```

```
                # Change PIN number
                case "0":
                    while (timesub(date(), time(), sdate,
stime) <= 20)
                        # clear digit buffer
                        sc_clrdigits(vpline) ;

                        # check for call waiting request
                        check_waiting(NOCONF) ;

                        # give PIN programming selections
                        voslog("sc_play('PIN
selection'(???).vox)") ;
                        sc_play(vpline, "pinselct.vox") ;

                        if ((code = sc_getdigits(vpline,
1, 2, 2)) eq T_MAXDT)
                            break;
                        endif
                    endwhile
                    # check for dropped out of entry loop
due to tone entry
                    if (code <> T_MAXDT)
                        break ;
                    endif

                    # check for call waiting request
                    check_waiting(NOCONF) ;

                    code = sc_digits(vpline) ;
                    voslog("Subscriber selection number:
"&code) ;

                    switch (code)

                    # Change programming PIN
                    case "1":
                        # clear digit buffer
                        sc_clrdigits(vpline) ;

                        # ask for new programming PIN
number
                        voslog("sc_play('enter
programming PIN'(???).vox)") ;
                        sc_play(vpline, "PRGRMPIN.vox") ;

                        # set '*' and '#' as mask digits
                        sc_toneint(vpline, 1, "*#") ;

                        if (((code = sc_getdigits(vpline,
16, 10, 3)) <> T_MAXDT) and (code <> T_MDTMF) and (code
```

```
<> T_SIL))
                                voslog("ERROR: "&code&" -
entering voice mail number failure") ;
                           else
                                phone_num = sc_digits(vpline)
;

                                # allow entry of a second '#'
or '*'
                                if (((code =
sc_getdigits(vpline, 1, 1, 1)) <> T_MAXDT) and (code <>
T_MDTMF) and (code <> T_SIL))
                                     voslog("ERROR: "&code&"
- entering voice mail number failure") ;
                                endif

                                phone_num =
phone_num&sc_digits(vpline) ;
                                voslog("Programming PIN:
"&phone_num) ;

                                # leave old number if '*'
entered
                                if (phone_num strneq "*")
                                     # remove any terminating
'*' or '#'
                                     if ((substr(phone_num,
length(phone_num), 1) streq "*")
                                      or (substr(phone_num,
length(phone_num), 1) streq "#"))
                                          phone_num =
substr(phone_num, 1, length(phone_num) - 1) ;
                                endif

                                # update subscriber's
programming PIN
                                glb_set(PRGRM_PIN+subs,
phone_num) ;

                                # open programming PIN
file
                                file =
fil_open("program"&subs&".nmb", "wcts") ;

                                # seek start of file
                                fil_seek(file, 0, 0) ;

                                # write new programming
PIN string to file
                                fil_writes(file,
glb_get(PRGRM_PIN+subs)&"`r`n") ;
```

```
                                    # close file
                                    fil_close(file) ;
                               endif

                               # indicate current
programming PIN follows
                               voslog("sc_play('program PIN
is'(????).vox)") ;
                               sc_play(vpline,
"PRGRMPIS.vox") ;

                               # play current programming
PIN
                               if (glb_get(PRGRM_PIN+subs)
strneq "")
                                    voslog("spk_dgts('
"&glb_get(PRGRM_PIN+subs)&"')") ;
                                    spk_dgts(glb_get
(PRGRM_PIN+subs)) ;
                               else
                                    voslog("sc_play('null'
(????).vox)") ;
                                    sc_play(vpline,
"null.vox") ;
                               endif
                          endif

                          # enable tone interruption
                          sc_toneint(vpline, 1) ;

                     # Change barge in PIN
                     case "2":
                          # clear digit buffer
                          sc_clrdigits(vpline) ;

                          # ask for new barge in PIN number
                          voslog("sc_play('enter barge in
PIN'(???).vox)") ;
                          sc_play(vpline, "BARGEPIN.vox") ;

                          # set '*' and '#' as mask digits
                          sc_toneint(vpline, 1, "*#") ;

                          if (((code = sc_getdigits(vpline,
16, 10, 3)) <> T_MAXDT) and (code <> T_MDTMF) and (code
<> T_SIL))
                               voslog("ERROR: "&code&" -
entering voice mail number failure") ;
                          else
                               phone_num = sc_digits(vpline)
```

```
;

                              # allow entry of a second '#'
or '*'
                              if (((code =
sc_getdigits(vpline, 1, 1, 1)) <> T_MAXDT) and (code <>
T_MDTMF) and (code <> T_SIL))
                                   voslog("ERROR: "&code&"
- entering forwarding number failure") ;
                              endif

                              phone_num =
phone_num&sc_digits(vpline) ;
                              voslog("Barge In PIN:
"&phone_num) ;

                              # check for leaving old
number
                              if (phone_num strneq "*")
                                   # remove any terminating
'*' or '#'
                                   if ((substr(phone_num,
length(phone_num), 1) streq "*")
                                    or (substr(phone_num,
length(phone_num), 1) streq "#"))
                                        phone_num =
substr(phone_num, 1, length(phone_num) - 1) ;
                                   endif

                                   # update subscriber's
barge in PIN
                                   glb_set(BARGE_PIN+subs,
phone_num) ;

                                   # open programming PIN
file
                                   file =
fil_open("bargein"&subs&".nmb", "wcts") ;

                                   # seek start of file
                                   fil_seek(file, 0, 0) ;

                                   # write new barge in PIN
string to file
                                   fil_writes(file,
glb_get(BARGE_PIN+subs)&"`r`n") ;

                                   # close file
                                   fil_close(file) ;
                              endif
```

```
PIN follows                         # indicate current barge in
is'(????).vox)") ;                  voslog("sc_play('barge in PIN
"BARGEPIS.vox") ;                   sc_play(vpline,

                                    # play current barge in PIN
strneq "")                          if (glb_get(BARGE_PIN+subs)
"&glb_get(BARGE_PIN+subs)&"')") ;       voslog("spk_dgts('
(BARGE_PIN+subs)) ;                     spk_dgts(glb_get
                                    else
(????).vox)") ;                         voslog("sc_play('null'
"null.vox") ;                           sc_play(vpline,
                                    endif
                            endif

                            # enable tone interruption
                            sc_toneint(vpline, 1) ;

                        endswitch

                    # Transfer to voice mail
                    case "3":
                    case_3:

                        phone_num = glb_get(VOICEMAIL+subs);
                        goto out_bound ;

                    case "9":
                    # TEST - outbound calling function
                    if (glb_get(CALLER_DID+subs) streq
"6545786")
                        # play dial tone message
                        voslog("sc_play('dial
tone'(274).vox)") ;
                        sc_play(vpline, "PR0274.vox") ;

                        # set '*' and '#' as mask digits
                        sc_toneint(vpline, 1, "#") ;

                        if (((code = sc_getdigits(vpline, 7,
10, 3)) <> T_MAXDT) and (code <> T_MDTMF) and (code <>
T_SIL))
                            voslog("ERROR: "&code&" -
entering outbound number failure") ;
```

```
                    goto sel_loop ;
               else
                    phone_num = sc_digits(vpline) ;
                    # remove any terminating '*' or
'#'
                    if ((substr(phone_num,
length(phone_num), 1) streq "*")
                     or (substr(phone_num,
length(phone_num), 1) streq "#"))
                         phone_num = substr(phone_num,
1, length(phone_num) - 1) ;
                    endif
               endif

               # enable tone interruption
               sc_toneint(vpline, 1) ;
          out_bound:
               # set outbound call number
               if (glb_get(CALL1ACT+subs) eq 0)
                    # call #1 inactive - so use call
#1
                    call = 0 ;
               else
                    if (glb_get(CALL2ACT+subs) eq 0)
                         # call #2 inactive - so use
call #2
                         call = 1 ;
                    else
                         voslog("ERROR: No barger
calls available") ;
                         call = "" ;
                         goto sel_loop ;
                    endif
               endif

               if (phone_num streq
glb_get(VOICEMAIL+subs))
                    # ask outbound alternate netline
be allocated
                    msg_put(mainpid, "A"&vpline) ;
                    if ((code = msg_get(3)) strneq
"A-OK")
                         voslog("ERROR: code "&code&"
when allocating outbound netline") ;
                         call = "" ;
                         goto sel_loop ;
                    endif

                    # get allocated outbound
alternate netline
                    while (length(outline =
```

```
msg_get(3)) eq 0)
                        endwhile
                        if ((outline < ALTBGN) or
(outline > ALTEND))
outline number "&outline&"  voslog("ERROR: invalid
                            message") ;
                            call = "" ;
                            goto sel_loop;
                        endif
                    else
                        # ask outbound netline be
allocated
                        msg_put(mainpid, "N"&vpline) ;
                        if ((code = msg_get(3)) strneq
"N-OK")
                            voslog("ERROR: code "&code&"
when allocating outbound netline") ;
                            call = "" ;
                            goto sel_loop ;
                        endif

                        # get allocated outbound netline
                        while (length(outline =
msg_get(3)) eq 0)
                        endwhile
                        if ((outline < NETBGN) or
(outline > NETEND))
                            voslog("ERROR: invalid
outline number "&outline&"  message") ;
                            call = "" ;
                            goto sel_loop ;
                        endif
                    endif

                    # ask conference be allocated
                    msg_put(mainpid, "C") ;
                    if ((code = msg_get(3)) strneq "C-OK")
                        voslog("ERROR: code "&code&" when
allocating conference") ;
                        call = "" ;
                        goto sel_loop ;
                    endif

                    # get allocated conference number
                    while (length(conf = msg_get(3)) eq 0)
                    endwhile
                    if (conf < 1 or conf > CONFMAX)
                        voslog("ERROR: invalid conference
number "&conf&" message") ;
                        call = "" ;
                        goto sel_loop ;
```

```
                    endif

                    voslog("Allocating conference "&conf)
;

# temporarily wrong vpline # set outbound call active
                    glb_set(CALL1ACT+SUBSMAX*call+subs,
vpline) ;

# temporarily wrong vpline # set outbound call answered
                    glb_set(CALL1ANS+SUBSMAX*call+subs,
vpline) ;

                    # set outbound conference number
                    glb_set(CALL1CNF+SUBSMAX*call+subs,
conf) ;

                    # set outbound caller netline
                    glb_set(CALL1NET+SUBSMAX*call+subs,
outline) ;

                    # set outbound barger active
                    glb_set(BARG1SUB+SUBSMAX*call+subs,
netline) ;

                    # set outbound barger answered
                    glb_set(BARG1ANS+SUBSMAX*call+subs,
vpline) ;

                    # set outbound forwarding inactive
 (not started)
                    glb_set(FWRD1ACT+SUBSMAX*call+subs,
 "") ;

                    # set outbound forwarding unanswered
                    glb_set(FWRD1ANS+SUBSMAX*call+subs,
 "") ;

                    # set outbound forwarding netline
 inactive
                    glb_set(FWRD1NET+SUBSMAX*call+subs,
 "") ;

                    # set outbound forwarding call
 progress inactive
                    glb_set(FWRD1CPA+SUBSMAX*call+subs,
 "") ;

                    # set outbound voice mail inactive
 (not started)
                    glb_set(MAIL1ACT+SUBSMAX*call+subs,
```

```
"") ;

                    # set outbound voice mail netline
inactive
                    glb_set(MAIL1NET+SUBSMAX*call+subs,
"") ;

                    # ignore disconnects
                    sc_watch(vpline, "lw+-", 1) ;

                    # drive vpline receive from outline
transmit
                    if ((code = Scon(vpline, outline,
SSAS_PASS, SSAL_NO)) <> 0)
                        voslog("Scon("&vpline&",
"&outline&") code "&code&" error") ;
                    else
                        voslog("Scon("&vpline&",
"&outline&"): "&Sstatus(vpline)) ;
                    endif

                    # drive outline receive from vpline
transmit
                    if ((code = Scon(outline, vpline,
SSAS_PASS, SSAL_NO)) <> 0)
                        voslog("Scon("&outline&",
"&vpline&") code "&code&" error") ;
                    else
                        voslog("Scon("&outline&",
"&vpline&"): "&Sstatus(outline)) ;
                    endif

                    # save the initial time
                    sdate = date() ;
                    stime = time() ;
          out_seize:
                    # go off hook
                    if ((code = sc_offhook(vpline)) <>
T_OFFH)
                        voslog("ERROR: code "&code&"
while putting vpline "&vpline&" offhook") ;
                        goto out_retry ;
                    else
                        voslog("sc_offhook("&vpline&")")
;
                    endif

                    # wait one second for next event
report
                    code = sc_wait(vpline, 10) ;
                    voslog("sc_wait("&vpline&",10) =
```

```
"&code) ;

                    # check for loop current event report
                    if (code <> T_LCON)
                         voslog("ERROR: Loop current
timeout on vpline "&vpline) ;
                         goto out_retry ;
                    endif

                    # wait one second for next event
report
                    code = sc_wait(vpline, 10) ;
                    voslog("sc_wait("&vpline&",10) =
"&code) ;

                    # check for wink event report
                    if (code <> T_WKRECV)
                         voslog("ERROR: Wink timeout on
vpline "&vpline) ;
                         goto out_retry ;
                    endif

                    # drive netline receive from outline
transmit
                    if ((code = Scon(netline, outline,
SSAS_OFFHOOK, SSAL_NO)) <> 0)
                         voslog("Scon("&netline&",
"&outline&") code "&code&" error") ;
                    else
                         voslog("Scon("&netline&",
"&outline&"): "&Sstatus(netline)) ;
                    endif

                    voslog("Dialing outbound number
"&phone_num) ;

                    # dial the outbound number - wait
until dialing complete
                    if (phone_num streq
glb_get(VOICEMAIL+subs))
                         code = sc_dial(vpline,
glb_get(STRATUS_PRE)&phone_num&glb_get(STRATUS_PST)) ;
                    else
                         code = sc_dial(vpline,
glb_get(PSTN_PRE+subs)&phone_num&glb_get(PSTN_PST+subs))
;
                    endif
                    # check dialing completion coded
                    if (code <> T_DIAL)
                         voslog("ERROR: code "&code&"
while dialing outbound number "&phone_num) ;
```

```
        out_retry:
                        if (timesub(date(), time(),
sdate, stime) < 12)
                            # put vpline onhook
                            if ((code =
sc_onhook(vpline)) <> T_ONH)
                                voslog("ERROR: code
"&code&" while putting vpline "&vpline&" onhook") ;
                            else
                                voslog("sc_onhook
("&vpline&")") ;
                            endif

                            # wait for onhook recognition
                            sleep(10) ;

                            goto out_seize ;
                        else
        out_discon:
                            # release barger's outbound
call
                            release_outbound_call() ;

                            # drive vpline receive from
netline transmit
                            if ((code = Scon(vpline,
netline, SSAS_PASS, SSAL_NO)) <> 0)
                                voslog("Scon("&vpli
ne&","&netline&") code "&code&" error") ;
                            else
                                voslog("Scon("&vpli
ne&","&netline&"): "&Sstatus(vpline)) ;
                            endif

                            # drive netline receive from
vpline transmit
                            if ((code = Scon(netline,
vpline, SSAS_PASS, SSAL_YES)) <> 0)
                                voslog("Scon("&netl
ine&","&vpline&") code "&code&" error") ;
                            else
                                voslog("Scon("&netl
ine&","&vpline&"): "&Sstatus(netline)) ;
                            endif

                            # give time for netline loop
current to settle at vpline
                            sleep(3) ;

                            # allow disconnects to run
onsignal
```

```
                    sc_watch(vpline, "d+-", 1) ;

                    goto sel_loop;
                endif
            endif

            # drive vpline receive from netline
transmit
            if ((code = Scon(vpline, netline,
SSAS_PASS, SSAL_NO)) <> 0)
                voslog("Scon("&vpline&",
"&netline&") code "&code&" error") ;
            else
                voslog("Scon("&vpline&",
"&netline&"): "&Sstatus(vpline)) ;
            endif

            # drive netline receive from vpline
transmit
            if ((code = Scon(netline, vpline,
SSAS_PASS, SSAL_YES)) <> 0)
                voslog("Scon("&netline&",
"&vpline&") code "&code&" error") ;
            else
                voslog("Scon("&netline&",
"&vpline&"): "&Sstatus(netline));
            endif

            # give time for netline loop current
to settle at vpline
            sleep(3) ;

            # allow disconnects to run onsignal
            sc_watch(vpline, "d+-", 1) ;

            # add barger netline to outbound
conference
            add_conf(call, netline, LOGON) ;

            # add outbound netline to outbound
conference
            add_conf(call, outline, LOGON) ;

            voslog("Barger Wait for Outbound
Answer Loop") ;

            while (timesub(date(), time(), sdate,
stime) < 40)
                # clear digit buffer
                sc_clrdigits(vpline) ;
```

```
                    # check for call waiting request
                    check_waiting(TALKER) ;

                    # check for outbound answer
supervision
                    if (substr(Sstatus(outline), 10,
1) eq 1)
                        voslog("Answer supervison") ;
                        break ;
                    endif

                    # check for barger forced
disconnect
                    if (sc_getdigits(vpline, 1, 1, 1)
eq T_MAXDT)
                        if (sc_digits(vpline) streq
"#")
                            voslog("Forced
disconnect") ;
                            break ;
                        endif
                    endif
                endwhile
                # check for no outbound answer
supervision
                if (substr(Sstatus(outline), 10, 1) eq
0)
                    # release barger's voice mail
call
                    release_outbound_call() ;
                    goto sel_loop;
                endif

                # ask outbound CALLOUT task vpline be
allocated
                msg_put(mainpid,
"O"&call&outline&subs) ;
                code = msg_get(3) ;
                if (substr(code, 1, 1) strneq "O")
                    voslog("ERROR: code "&code&" when
allocating outbound vpline") ;
                    call = "" ;
                    # release barger's voice mail
call
                    release_outbound_call() ;
                    goto sel_loop ;
                endif

                # set outbound call active (to CALLOUT
vpline)
                glb_set(CALL1ACT+SUBSMAX*call+subs,
```

```
substr(code, 2, 99)) ;
                    # set outbound call answered (to
CALLOUT vpline)
                    glb_set(CALL1ANS+SUBSMAX*call+subs,
substr(code, 2, 99)) ;
                    # act as if have barged in
                    goto barge_loop ;
               else
                    # restart elapsed time
                    sdate = date() ;
                    stime = time() ;
               endif

               case "*":
                    if (sc_getdigits(vpline, 1, 1, 1) eq
T_MAXDT)
                         if (sc_digits(vpline) streq "*")
                              # check for a caller waiting
                              if ((glb_get(CALL1ACT+subs)
<> 0)
                                   or (glb_get(CALL2ACT+subs)
<> 0))
                                   goto barge_in ;
                              else
                                   voslog("sc_play(no call
waiting(108).vox") ;
                                   sc_play(vpline,
"PR0108.vox") ;
                              endif
                         endif
                    endif

               default:
                    # restart elapsed time
                    sdate = date() ;
                    stime = time() ;

               endswitch

               # remain in programming loop
               goto sel_loop ;
          endif

          # check to see if PIN entry matches barge in
password
          if (pin_entry streq glb_get(BARGE_PIN+subs))
               goto barge_in ;
          endif
     endwhile
```

```
     # dropped out of PIN entry loop due to timeout
     voslog("sc_play(goodbye(039).vox") ;
     sc_play(vpline, "PR0039.vox") ;
     hangup() ;
     chain("GETDID", vpline) ;

barge_in:
     # check for call number not set yet
     if (call streq "")
          # call number not set
          # check for call #1 active
          if (glb_get(CALL1ACT+subs) <> 0)
               # proceed to barge into call #1
               call = 0 ;
          else
               # check for caller #2 active
               if (glb_get(CALL2ACT+subs) <> 0)
                    # proceed to barge into call #2
                    call = 1 ;
               else
                    voslog("sc_play(no call
waiting(108).vox") ;
                    sc_play(vpline, "PR0108.vox") ;
                    goto get_pin ;
               endif
          endif
     else
          # call number already set
          # check for both calls active
          if ((glb_get(CALL1ACT+subs) <> 0)
          and (glb_get(CALL2ACT+subs) <> 0))
               # change to unselected call in case barger
backed out
               # from selected call during earlier
monitor barge in
               voslog("TEST: call switch from: "&call&"
to: "&modulo2(call));
               call = modulo2(call) ;
          endif
     endif

     # TEST - monitor barge function
     if (glb_get(CALLER_DID+subs) streq "6545786")
          # check for null forwarding number
          if (glb_get(FORWARD+subs) streq "")
               # shorten number of simulated rings
               # before caller is transfered to voice
mail

               # request answer supervision be returned
```

```
to caller
            # set negative to indicate monitor barger
request
            glb_set(CALL1ANS+SUBSMAX*call+subs, -
vpline) ;
        endif

        # reset call waiting request
        glb_set(CALLWAIT+subs, "") ;

        # add our netline as listener to selected
caller's conference
        add_list(call, netline, LOGON) ;

        for (;;)
            # clear digit buffer
            sc_clrdigits(vpline) ;

            # delete our netline as listener
            del_list(call, netline, vpline, LOGOFF) ;

            # check for call waiting request
            # check_waiting(LISTENER) ;     # net
effect
            # del_list & add_list already being done
for beep
            check_waiting(NOCONF) ;

            # play monitor mode beep to listener
            sc_playtone(vpline, 1000, 2000, -30, -30,
20) ;

            # add our netline as listener
            add_list(call, netline, LOGOFF) ;

            # space monitor beeps every 5 seconds
while looking for tone input
            if ((code = sc_getdigits(vpline, 1, 5, 1))
eq T_MAXDT)
                if ((code = sc_digits(vpline)) streq
"#")
                    # delete our netline as listener
to
                    # selected caller's conference
                    del_list(call, netline, vpline,
LOGON) ;

                    # restart elapsed time
                    sdate = date() ;
                    stime = time() ;
```

```
                    goto pin_loop ;
                endif
                # check for non-monitor barge request
and caller active
                if ((code streq "*")
                and
(glb_get(CALL1ACT+SUBSMAX*call+subs) <> 0))
                    break;
                endif
            endif
        endfor

        # delete our netline as listener to selected
caller's conference
        del_list(call, netline, vpline, LOGON) ;
    endif

    # set one barger call active
    glb_set(BARG1SUB+SUBSMAX*call+subs, netline) ;
    glb_set(BARG1SUB+SUBSMAX*modulo2(call)+subs, "") ;

    # set one barger call answered
    glb_set(BARG1ANS+SUBSMAX*call+subs, vpline) ;
    glb_set(BARG1ANS+SUBSMAX*modulo2(call)+subs, "") ;

    # stop caller's forwarding
    stop_forwarding(call, 0) ;

    # stop caller's voice mail
    stop_voicemail(call) ;

    # request answer supervision be returned to caller
    glb_set(CALL1ANS+SUBSMAX*call+subs, vpline) ;

    # abort caller's possible sc_play in progress
    sc_abort(glb_get(CALL1ACT+SUBSMAX*call+subs)) ;
    voslog("sc_abort("&glb_get(CALL1ACT+SUBSMAX*ca
ll+subs)&")") ;

    # add our vpline to conference
    add_conf(call, vpline, LOGOFF) ;

    # play joining conference tones
    sc_playtone(vpline, 1000, 2000, -30, -30, 100) ;
    voslog("sc_playtone(join conf)") ;

    # delete our vpline from conference
    del_conf(call, vpline, netline, LOGOFF) ;

    # add our netline to selected caller's conference
    add_conf(call, netline, LOGON) ;
```

```
    # reset call waiting request
    glb_set(CALLWAIT+subs, "") ;

barge_loop:
    voslog("Barger Conversation Loop") ;

    for (;;)
        # check for call waiting request
        check_waiting(TALKER) ;

        # check for selected caller active
        if (glb_get(CALL1ACT+SUBSMAX*call+subs) <> 0)
            # selected caller active
            # check for unselected caller inactive
            if
(glb_get(CALL1ACT+SUBSMAX*modulo2(call)+subs) eq 0)
                # selected caller active/unselected
caller inactive
                # set unselected subscriber inactive
                glb_set(BARG1SUB+SUBSMAX*modulo2
(call)+subs, "") ;
                # check for three way conference
                if ((glb_get(BARG1SUB+subs) eq 0)
                 or (glb_get(BARG2SUB+subs) eq 0))
                    # three way conference not active
                    # release unselected caller's
conference if unused
                    rel_conf(modulo2(call)) ;
                endif
            endif
        else
            # selected caller inactive
            # check for unselected caller inactive
            if
(glb_get(CALL1ACT+SUBSMAX*modulo2(call)+subs) eq 0)
                # both callers inactive
                # set both calls inactive but with
barger active
                glb_set(BARG1SUB+subs, 0) ;
                glb_set(BARG2SUB+subs, 0) ;
                # release selected caller's conference
if unused
                rel_conf(call) ;
                # release unselected caller's
conference if unused
                rel_conf(modulo2(call)) ;
            else
                # selected call inactive/unselected
call active
                # check for three way conference
```

```
                    if ((glb_get(BARG1SUB+subs) eq 0)
                     or (glb_get(BARG2SUB+subs) eq 0))
                         # three way conference not active
                         # do NOT set selected subscriber
inactive here

                         voslog("TEST: Auto switch
conference request") ;

                         # switch to unselected call
                         flash = "**" ;
                         goto switch_conf ;
                    else
                         # three way conference active
                         # set selected subscriber
inactive
                         glb_set(BARG1SUB+SUBSMAX*cal
l+subs, "") ;
                         # release selected caller's
conference if unused
                         rel_conf(call) ;

                         voslog("TEST: Call "&call&"
switched to become call "&modulo2(call)) ;
                         # switch to active (i.e.
unselected) call
                         call = modulo2(call) ;
                    endif
               endif
          endif

          # check for barger tone input
          sc_clrdigits(vpline) ;
          if ((code = sc_getdigits(vpline, 2, 3, 2)) eq
T_MAXDT)
               flash = sc_digits(vpline) ;

               # check for forced forwarding disconnect
request
               if (flash streq "#8")
                    # stop forwarding even if answered
                    stop_forwarding(call, 1) ;
               endif

               # check for forced voice mail request
               if (flash streq "#9")
                    # check for caller active
                    if
(glb_get(BARG1SUB+SUBSMAX*call+subs) strneq "")
                         stop_forwarding(call, 0) ;
                         # start voicemail with
```

```
announcement
                    start_voicemail(call, 1) ;
                endif
            endif
switch_conf:
            # check for conference switch request
            if (flash streq "***")
                # check for unselected call active
                #  and for three way conference not
active
                if
((glb_get(CALL1ACT+SUBSMAX*modulo2(call)+subs) <> 0)
                and ((glb_get(BARG1SUB+subs) eq 0)
                 or  (glb_get(BARG2SUB+subs) eq 0)))
                    voslog("Switch conference
request") ;

                    # reset call waiting request
                    glb_set(CALLWAIT+subs, "") ;

                    # delete our netline from
selected caller's conference
                    del_conf(call, netline, vpline,
LOGON) ;

                    # set new selected caller
                    call = modulo2(call) ;

                    # set selected barger call active
                    glb_set(BARG1SUB+SUBSMAX*cal
1+subs, netline) ;
                    # set unselected barger call
inactive
                    glb_set(BARG1SUB+SUBSMAX
*modulo2(call)+subs, "") ;

                    # release unselected conference
if unused
                    rel_conf(modulo2(call)) ;

                    # stop selected call's forwarding
                    stop_forwarding(call, 0) ;

                    # (FUTURE PROGRAMMABLE OPTION)
                    # stop selected call's voice mail
                    stop_voicemail(call) ;

                    # request answer supervision be
returned to selected caller
                    glb_set(CALL1ANS+SUBSMAX*cal
1+subs, vpline) ;
```

```
                        # set selected barger call
answered
                        glb_set(BARG1ANS+SUBSMAX*cal
l+subs, vpline) ;

                        # abort selected caller's
possible sc_play in progress
                        sc_abort(glb_get(CALL1ACT
+SUBSMAX*call+subs)) ;
                        voslog("sc_abort("&glb_get
(CALL1ACT+SUBSMAX*call+subs)&")") ;

                        # add our vpline to conference
                        add_conf(call, vpline, LOGOFF) ;

                        # play joining conference tones
                        sc_playtone(vpline, 1000, 2000, -
30, -30, 100) ;
                        voslog("sc_playtone(join conf)")
;

                        # delete our vpline from
conference
                        del_conf(call, vpline, netline,
LOGOFF) ;

                        # add our netline to selected
caller's conference
                        add_conf(call, netline, LOGON) ;
                    endif
                endif
                # check for conference switch request
                if (flash streq "#3")
                    # check for both callers active
                    #  and for three way conference not
active
                    if
((glb_get(CALL1ACT+SUBSMAX*call+subs) <> 0)
                    and
(glb_get(CALL1ACT+SUBSMAX*modulo2(call)+subs) <> 0)
                    and ((glb_get(BARG1SUB+subs) eq 0)
                     or  (glb_get(BARG2SUB+subs) eq 0)))
                        voslog("Three way conference
request") ;

                        # reset call waiting request
                        glb_set(CALLWAIT+subs, "") ;

                        # delete unselected caller's
netline from unselected caller's conference
```

```
                    del_conf(modulo2(call),
glb_get(CALL1NET+SUBSMAX*modulo2(call)+subs),
glb_get(CALL1ACT+SUBSMAX*modulo2(call)+subs), LOGON) ;

                    # request answer supervision be
returned to both callers
                    glb_set(CALL1ANS+SUBSMAX*cal
l+subs, vpline) ;
                    glb_set(CALL1ANS+SUBSMAX
*modulo2(call)+subs, vpline) ;

                    # stop unselected call's
forwarding
                    stop_forwarding(modulo2(call), 0)
;

                    # (FUTURE PROGRAMMABLE OPTION)
                    # stop selected call's voice mail
                    stop_voicemail(modulo2(call)) ;

                    # get unselected caller's
conference number
                    conf =
glb_get(CALL1CNF+SUBSMAX*modulo2(call)+subs) ;

                    # check for using active
specified conference
                    if ((conf > 0) and (conf <=
CONFMAX))
                         # check if conference still
in use
                         if
((glb_get(FWRD1SUB+SUBSMAX*modulo2(call)+subs) eq 0)
                         and
(glb_get(FWRD1ACT+SUBSMAX*modulo2(call)+subs) eq 0)
                         and
(glb_get(MAIL1ACT+SUBSMAX*modulo2(call)+subs) eq 0))
                              # request conference
deallocation
                              msg_put(mainpid,
"E"&conf) ;
                              if ((code = msg_get(3))
strneq "E-OK")
                                   voslog("ERROR: code
"&code&" when deallocating conference "&conf) ;
                              else
                                   voslog("Deallocating
conference "&conf) ;
                                   # set unselected
caller's conference inactive
                                   glb_set(CALL1CN
```

```
F+SUBSMAX*modulo2(call)+subs, "") ;
                                endif
                            endif
                        else
                            voslog("ERROR: deallocating
inactive conference") ;
                        endif

                        # set both barger calls active
                        glb_set(BARG1SUB+subs, netline) ;
                        glb_set(BARG2SUB+subs, netline) ;

                        # set both barger calls answered
                        glb_set(BARG1ANS+subs, vpline) ;
                        glb_set(BARG2ANS+subs, vpline) ;

                        # abort unselected caller's
possible sc_play in progress
                        sc_abort(glb_get(CALL1ACT
+SUBSMAX*modulo2(call)+subs)) ;
                        voslog("sc_abort("&glb_get
(CALL1ACT+SUBSMAX*modulo2(call)+subs)&")") ;

                        # add our vpline to conference
                        add_conf(call, vpline, LOGOFF) ;

                        # play joining conference tones
                        sc_playtone(vpline, 1000, 2000, -
30, -30, 100) ;
                        voslog("sc_playtone(join conf)")
;

                        # delete our vpline from
conference
                        del_conf(call, vpline, netline,
LOGOFF) ;

                        # add unselected caller's netline
to selected caller's conference
                        add_conf(call,
glb_get(CALL1NET+SUBSMAX*modulo2(call)+subs), LOGON) ;

                        # check for unselected conference
inactive
                        if
(glb_get(CALL1CNF+SUBSMAX*modulo2(call)+subs) strneq "")
                            voslog("TEST: Deselected
conference
"&glb_get(CALL1CNF+SUBSMAX*modulo2(call)+subs)&" not
inactive") ;
                        endif
```

```
                    # update unselected caller's
conference to selected caller's conference
                    glb_set(CALL1CNF+SUBSMAX
*modulo2(call)+subs,
glb_get(CALL1CNF+SUBSMAX*call+subs)) ;
                endif
            endif
        endif

        # check to see if barger is talking to anyone
        if ((glb_get(CALL1ACT+subs) eq 0)
        and (glb_get(CALL2ACT+subs) eq 0)
        and (glb_get(FWRD1ACT+subs) eq 0)
        and (glb_get(FWRD2ACT+subs) eq 0))
            # delete our vpline from our conference
            del_conf(call, vpline, netline, LOGOFF) ;

            # delete our netline from our conference
            del_conf(call, netline, vpline, LOGON) ;

            # drive netline receive from vpline
transmit
            if ((code = Scon(netline, vpline,
SSAS_PASS, SSAL_NO)) <> 0)
                voslog("Scon("&netline&","&vpline&")
code "&code&" error") ;
            else
                voslog("Scon("&netline&","&vpline&"):
"&Sstatus(netline)) ;
            endif

            # drive vpline receive from netline
transmit
            if ((code = Scon(vpline, netline,
SSAS_PASS, SSAL_NO)) <> 0)
                voslog("Scon("&vpline&","&netline&")
code "&code&" error") ;
            else
                voslog("Scon("&vpline&","&netline&"):
"&Sstatus(vpline)) ;
            endif

            # set both calls inactive but with barger
present
            glb_set(BARG1SUB+subs, 0) ;
            glb_set(BARG2SUB+subs, 0) ;

            # release selected caller's conference if
unused
            rel_conf(call) ;
```

```
                # release unselected caller's conference
if unused
                rel_conf(modulo2(call)) ;

                # set call number inactive
                call = "" ;

                if (pin_entry streq
glb_get(PRGRM_PIN+subs))
                     # go back to programming selection
loop
                     goto sel_loop ;
                else
                     # start PIN entry loop elapsed time
                     sdate = date() ;
                     stime = time() ;

                     # go back into PIN entry loop
                     goto pin_loop ;
                endif
           endif
     endfor
end

#-----------------------------------------------------------
----------------------

onsignal
     voslog("Onsignal") ;
     hangup() ;
     chain("GETDID", vpline) ;
end

func hangup()
     # abort our multitasking calls in progress
     sc_abort(vpline) ;

     # wait for all our multitasking calls to finish
     for (code = 1; (sc_stat(vpline) <> 0) and (code <=
20); code++)
          sleep(5) ;
     endfor
     if (sc_stat(vpline) <> 0)
          voslog("ERROR: 10 seconds on vpline "&vpline&"
without idle status") ;
     endif

     # put vpline onhook
     while ((code = sc_onhook(vpline)) <> T_ONH)
          voslog("ERROR: code "&code&" while putting
```

```
vpline "&vpline&" onhook") ;
          sleep(10) ;
     endwhile
     voslog("sc_onhook("&vpline&")") ;

     # delete barger vpline from our conference
     del_conf(call, vpline, netline, LOGOFF) ;

     # delete barger netline from our conference
     del_conf(call, netline, vpline, LOGON) ;

     # remove vpline receive from netline transmit
     if ((code = Sbrk(vpline, netline, SSAS_ONHOOK,
SSAL_NO)) <> 0)
          voslog("Sbrk("&vpline&","&netline&") code
"&code&" error") ;
     else
          voslog("Sbrk("&vpline&","&netline&"):
"&Sstatus(vpline)) ;
     endif

     if ((netline >= NETBGN) and (netline <= NETEND))
          # remove netline receive from vpline transmit
          # and enable netline SS96 alerting
          if ((code = Sbrk(netline, vpline, SSAS_ONHOOK,
SSAL_YES)) <> 0)
               voslog("Sbrk("&netline&","&vpline&") code
"&code&" error") ;
          else
               voslog("Sbrk("&netline&","&vpline&"):
"&Sstatus(netline)) ;
          endif
     else
          # remove netline receive from vpline transmit
          # and disable netline SS96 alerting
          if ((code = Sbrk(netline, vpline, SSAS_ONHOOK,
SSAL_NO)) <> 0)
               voslog("Sbrk("&netline&","&vpline&") code
"&code&" error") ;
          else
               voslog("Sbrk("&netline&","&vpline&"):
"&Sstatus(netline)) ;
          endif
     endif

     # release netline/vpline allocations
     dealloc() ;

     # set both barger calls inactive
     glb_set(BARG1SUB+subs, "") ;
     glb_set(BARG2SUB+subs, "") ;
```

```
     # release our conference if unused
     rel_conf(call) ;

     # set task debug screen status
     scr_stat("Idle") ;
endfunc

func dealloc()
     if (netline strneq "")
          # release netline/vpline allocations
          msg_put(mainpid, "D"&netline&vpline) ;
          if ((code = msg_get(3)) strneq "D-OK")
               voslog("ERROR: code "&code&" when
deallocating netline "&netline) ;
          endif
     endif
endfunc

func check_waiting(sconf)
     # check for call waiting request
     if (glb_get(CALLWAIT+subs) <> 0)
          if (sconf eq TALKER)
               # delete our netline as talker from
conference
               del_conf(call, netline, vpline, LOGOFF) ;
          else
               if (sconf eq LISTENER)
                    # delete our netline as listener from
conference
                    del_list(call, netline, vpline,
LOGOFF) ;
               endif
          endif
          # check for priority call waiting request
          if (glb_get(CALLWAIT+subs) < 0)
               voslog("sc_play('urgent call waiting
tones'(1304).vox)") ;
               sc_play(vpline, "PR1304.vox") ; # two tones
               sc_play(vpline, "PR1304.vox") ; # two tones
          else
               voslog("sc_play('call waiting
tones'(1305).vox)") ;
               sc_play(vpline, "PR1305.vox") ; # three
tones
          endif
          if (sconf eq TALKER)
               # add our netline as talker to conference
               add_conf(call, netline, LOGOFF) ;
          else
               if (sconf eq LISTENER)
```

```
                    # add our netline as listener to
conference
                    add_list(call, netline, LOGOFF) ;
                endif
            endif
            # reset call waiting request
            glb_set(CALLWAIT+subs, "") ;
        endif
endfunc

func release_outbound_call()

    # check for outbound call active
    if (call strneq "")
        voslog("release_outbound_call()") ;

        # delete barger vpline from outbound conference
        del_conf(call, vpline, netline, LOGOFF) ;

        # delete barger netline from outbound
conference
        del_conf(call, netline, vpline, LOGON) ;

        # delete outbound netline from conference (no
vpline)
        del_conf(call, outline, 0, LOGON);

        # set outbound conference inactive (but leave
number usable)
        glb_set(CALL1CNF+SUBSMAX*call+subs,
"+"&glb_get(CALL1CNF+SUBSMAX*call+subs)) ;

        # set outbound netline inactive
        glb_set(CALL1NET+SUBSMAX*call+subs, "") ;

        # set outbound answer supervision inactive
        glb_set(CALL1ANS+SUBSMAX*call+subs, "") ;

        # set outbound caller inactive (do last)
        glb_set(CALL1ACT+SUBSMAX*call+subs, "") ;

        if ((outline >= NETBGN) and (outline <=
NETEND))
            # remove outline receive from vpline
transmit
            # and enable outline SS96 alerting
            if ((code = Sbrk(outline, vpline,
SSAS_ONHOOK, SSAL_YES)) <> 0)
                voslog("Sbrk("&outline&","&vpline&")
code "&code&" error") ;
            else
```

```
                    voslog("Sbrk("&outline&","&vpline&"):
"&Sstatus(outline)) ;
               endif
          else
               # remove outline receive from vpline
transmit
               # and disable outline SS96 alerting
               if ((code = Sbrk(outline, vpline,
SSAS_ONHOOK, SSAL_NO)) <> 0)
                    voslog("Sbrk("&outline&","&vpline&")
code "&code&" error") ;
               else
                    voslog("Sbrk("&outline&","&vpline&"):
"&Sstatus(outline)) ;
               endif
          endif

          # release outline allocation (no vpline)
          msg_put(mainpid, "D"&outline&0) ;
          if ((code = msg_get(3)) strneq "D-OK")
               voslog("ERROR: code "&code&" when
deallocating outline "&outline) ;
          endif

          # release outbound conference if unused
          rel_conf(call) ;

          # set outbound call inactive
          call = "" ;
     endif
endfunc

func stop_forwarding(scall, sforce)

     # check for not forced stop request
     # and for forwarding answered
     # and for not "transparent" forwarding mode
     if ((sforce eq 0)
     and (glb_get(FWRD1ANS+SUBSMAX*scall+subs) <> 0)
     and (glb_get(FORWARD+subs) strneq
glb_get(CALLER_DID+subs)))
          return;
     endif
     sem_set(SEM_FWRD+subs) ;

     # check for forwarding active
     if (glb_get(FWRD1ACT+SUBSMAX*scall+subs) <> 0)
          voslog("stop_forwarding()") ;

          # delete forwarding netline from caller's
conference
```

```
        del_conf(scall,
glb_get(FWRD1NET+SUBSMAX*scall+subs),
glb_get(FWRD1ACT+SUBSMAX*scall+subs), LOGON) ;

        # set call answered so forward task marks
itself inactive and stopped
        glb_set(FWRD1ANS+SUBSMAX*call+subs, vpline) ;

        # abort forwarding task
        sc_hangup(glb_get(FWRD1ACT+SUBSMAX*scall+subs))
;
        voslog("sc_hangup("&glb_get(FWRD1ACT+SUBSM
AX*scall+subs)&")") ;

        # set forwarding task inactive and stopped
        glb_set(FWRD1ACT+SUBSMAX*scall+subs, 0) ;
    endif
    sem_clear(SEM_FWRD+subs) ;
endfunc

func start_voicemail(scall, smsg)

    # check for voice mail active
    if (glb_get(MAIL1ACT+SUBSMAX*scall+subs) eq 0)
        # request voice mail vpline task be started
        msg_put(mainpid, "V"&scall&subs) ;
        code = msg_get(3) ;
        if (substr(code, 1, 1) strneq "V")
            voslog("ERROR: code "&code&" when
allocating voice mail vpline") ;
        else
            # set voice mail task active
            glb_set(MAIL1ACT+SUBSMAX*scall+subs,
substr(code, 2, 99)) ;

            if (smsg <> 0)
                # add our vpline to conference
                add_conf(scall, vpline, LOGOFF) ;

                # play connecting call to voice mail
message
                voslog("sc_play('to voice
mail'(030).vox)") ;
                sc_play(vpline, "PR0030.vox") ;

                # delete our vpline from conference
                del_conf(scall, vpline, netline,
LOGOFF) ;
            endif
        endif
    endif
```

```
endfunc

func stop_voicemail(scall)

     sem_set(SEM_MAIL+subs) ;

     # check for voice mail active
     if (glb_get(MAIL1ACT+SUBSMAX*scall+subs) <> 0)
          voslog("stop_voicemail()") ;

          # delete voice mail netline from caller's
conference
          del_conf(scall,
glb_get(MAIL1NET+SUBSMAX*scall+subs),
glb_get(MAIL1ACT+SUBSMAX*scall+subs), LOGON) ;

          # abort voice mail task
          sc_hangup(glb_get(MAIL1ACT+SUBSMAX*scall+subs))
;
          voslog("sc_hangup("&glb_get(MAIL1ACT+SUBSM
AX*scall+subs)&")") ;

          # set voice mail task inactive and stopped
          glb_set(MAIL1ACT+SUBSMAX*scall+subs, 0) ;
     endif
     sem_clear(SEM_MAIL+subs) ;
endfunc

func modulo2(scall)
     return    ((scall+1)-(scall+1)/2*2) ;
endfunc

     include "confsubs.inc"      # conferencing
subroutines
```

```
#
#    CALLIN.VS - D121 line task
#
#    Designed to work in concert with GETDID.VS
#
#-------------------------------------------------------
-----------------------

dec
     include "datmod.inc"        # SS96 library
definitions
     include "global.inc"        # define global variable
numbers
```

```
    var code:16 ;           # return value from function
calls
    var vpline:3 ;              # voice processing line
number
    var netline:3 ;             # network line number
    var conf:3 ;            # conference number
    var subs:3 ;            # subscriber number
    var call:3 ;            # call number
    var mainpid:3 ;             # main task PID
    var sdate:7 ;           # call starting date
    var stime:7 ;           # call starting time
    var etime:7 ;           # call elapsed time
    var calltime:7 ;        # call elapsed time
    var nulltime:7 ;        # null forward number elapsed
time
    var pin_entry:10 ;          # barger PIN entry

    var stats:16 ;              # beginning netline
status (confsubs)
    var line:3 ;            # needed for scr_stat
end

program

# Initializations
    mainpid = glb_get(MAINPID) ;

    vpline = arg() ;
    if ((vpline < GDIDBGN) or (vpline > GDIDEND))
        voslog("ERROR: invalid vpline number "&vpline&"
message received") ;
        hangup() ;
        chain("GETDID", vpline) ;
    endif
    line = vpline ;                 # debug only

# Begin call processing
    # enable hang up jump to onsignal
    sc_use(vpline) ;

    # set call number inactive (in case of unexpected
disconnect)
    call = "" ;

    # do not allow disconnects to run onsignal
    sc_watch(vpline, "+-", 1) ;

    while (length(netline = msg_get(3)) eq 0)
    endwhile
    if ((netline < NETBGN) or (netline > NETEND))
        if ((netline < ALTBGN) or (netline > ALTEND))
```

```
                voslog("ERROR: invalid netline number
 "&netline&" message received") ;
                hangup() ;
                chain("GETDID", vpline) ;
           endif
      endif

      while (length(subs = msg_get(3)) eq 0)
      endwhile
      if ((subs < 0) or (subs >= SUBSMAX))
           voslog("ERROR: invalid subscriber number
 "&subs&" message received") ;
           hangup() ;
           chain("GETDID", vpline) ;
      endif

      # set task debug screen status
      scr_stat("I-"&netline) ;

      # for testing - do not disconnect when changing
 BARGER into CALLIN
      if (glb_get(CALLER_DID+subs) strneq "6545786")
           # check for caller disconnect (preceding answer
 supervison return = no "ringing")
           if (sc_stat(vpline, 8) eq 1)
                voslog("Disconnect - ringing drop") ;
                hangup() ;
                chain("GETDID", vpline) ;
           endif
      endif

      # set our call number (GETDID checked for all calls
 active)
      if (glb_get(CALL1ACT+subs) eq 0)
           # call #1 inactive - so this becomes call #1
           call = 0 ;
      else
           if (glb_get(CALL2ACT+subs) eq 0)
                # call #2 inactive - so this becomes call
 #2
                call = 1 ;
           else
                if (glb_get(CALL3ACT+subs) eq 0)
                     # call #3 inactive - so this becomes
 call #3
                     call = 2 ;
                else
                     if (glb_get(CALL4ACT+subs) eq 0)
                          # call #4 inactive - so this
 becomes call #4
                          call = 3 ;
```

```
                    else
                        voslog("ERROR: No inbound calls
available") ;
                        play_reorder("") ;
                        hangup() ;
                        chain("GETDID", vpline) ;
                    endif
                endif
            endif
        endif

        # set caller active
        glb_set(CALL1ACT+SUBSMAX*call+subs, vpline) ;

        # ask conference be allocated
        msg_put(mainpid, "C") ;
        if ((code = msg_get(3)) strneq "C-OK")
            voslog("ERROR: code "&code&" when allocating
conference") ;
            play_reorder("") ;
            hangup() ;
            chain("GETDID", vpline) ;
        endif

        # get allocated conference number
        while (length(conf = msg_get(3)) eq 0)
        endwhile
        if ((conf < 1) or (conf > CONFMAX))
            voslog("ERROR: invalid conference number
"&conf&" message") ;
            play_reorder("") ;
            hangup() ;
            chain("GETDID", vpline) ;
        endif

        voslog("Allocating conference "&conf) ;

        # set conference number
        glb_set(CALL1CNF+SUBSMAX*call+subs, conf) ;

        # set caller netline
        glb_set(CALL1NET+SUBSMAX*call+subs, netline) ;

        # set answer supervision inactive
        glb_set(CALL1ANS+SUBSMAX*call+subs, "") ;

        if (call < BARGMAX)
            # set barger answered inactive
            glb_set(BARG1ANS+SUBSMAX*call+subs, "") ;
        endif
```

```
    if (call < FWRDMAX)
        # set forwarding inactive (not started)
        glb_set(FWRD1ACT+SUBSMAX*call+subs, "") ;

        # set forwarding answered inactive
        glb_set(FWRD1ANS+SUBSMAX*call+subs, "") ;

        # set forwarding netline inactive
        glb_set(FWRD1NET+SUBSMAX*call+subs, "") ;

        # set forwarding call progress inactive
        glb_set(FWRD1CPA+SUBSMAX*call+subs, "") ;
    endif

    if (call < MAILMAX)
        # set voice mail inactive (not started)
        glb_set(MAIL1ACT+SUBSMAX*call+subs, "") ;

        # set voice mail netline inactive
        glb_set(MAIL1NET+SUBSMAX*call+subs, "") ;
    endif

    # now allow disconnects to run onsignal
    sc_watch(vpline, "d+-", 1) ;

    # return answer supervision
    while ((code = sc_offhook(vpline)) <> T_OFFH)
        voslog("ERROR: code "&code&" while taking
vpline "&vpline&" offhook") ;
        sleep(10) ;
    endwhile
    voslog("Answer supervision") ;

    # clear digit buffer
    sc_clrdigits(vpline) ;

    # check for transparent forwarding
    if (glb_get(FORWARD+subs) streq
glb_get(CALLER_DID+subs))
        # start paging task
        start_paging();
        goto short_cut ;
    endif

    # play ringing message (allows tone interruption)
    voslog("sc_play('ringing'(PR0001).vox)") ;
    if ((code = sc_play(vpline, "PR0001.vox")) eq
T_TERMDT)
        if ((code = check_entry()) eq 1)
            goto voice_mail ;
        endif
```

```
        if (code eq 2)
             # TEST - reverse action of '0' for Ron
Brooks DID
             if (glb_get(CALLER_DID+subs) streq
"6545786")
                  goto start_page ;
             else
                  goto short_cut ;
             endif
        endif
    endif

    # TEST - reverse action of '0' for Ron Brooks DID
    if (glb_get(CALLER_DID+subs) streq "6545786")
        goto short_cut ;
    endif

start_page:

    # start paging task
    start_paging();

    # check for no early barger active
    if ((call >= BARGMAX) or
(glb_get(BARG1SUB+SUBSMAX*call+subs) eq 0))
        # play 'reached Accessline for' message
        voslog("sc_play('Accessline'(PR0601).vox)") ;
        if ((code = sc_play(vpline, "PR0601.vox")) eq
T_TERMDT)
             # check for termination by caller
             if ((code = check_entry()) eq 1)
                  goto voice_mail ;
             endif
             if (code eq 2)
                  goto short_cut ;
             endif
        endif
    endif

    # check for no early barger active
    if ((call >= BARGMAX) or
(glb_get(BARG1SUB+SUBSMAX*call+subs) eq 0))
        # play subscriber's name message
        voslog("sc_play('subscriber
name'("&glb_get(CALLER_DID+subs)&").nam)") ;
        if ((code = sc_play(vpline,
glb_get(CALLER_DID+subs)&".nam")) eq T_TERMDT)
             # check for termination by caller
             if ((code = check_entry()) eq 1)
                  goto voice_mail ;
             endif
```

```
                if (code eq 2)
                     goto short_cut ;
                endif
           endif
      endif

      # check for no early barger active
      if ((call >= BARGMAX) or
(glb_get(BARG1SUB+SUBSMAX*call+subs) eq 0))
           # play 'if busy or unanswered' message
           voslog("sc_play('if unanswered'(D008).vox)") ;
           if ((code = sc_play(vpline, "D008.vox")) eq
T_TERMDT)
                # check for termination by caller
                if ((code = check_entry()) eq 1)
                     goto voice_mail ;
                endif
           endif
      endif

short_cut:
      # check for no early barger active (barger's
sc_abort will stop
      # playing message, but will not stop these tasks
from being started)
      if ((call >= BARGMAX) or
(glb_get(BARG1SUB+SUBSMAX*call+subs) eq 0))
           # check for not null forwarding number
           if (glb_get(FORWARD+subs) strneq "")
                # check for existance of subscriber
forwarding task
                if (call < FWRDMAX)
                     # start forwarding
                     start_forwarding(call) ;
                else
voice_mail:
                     # start voice mail without
announcement
                     start_voicemail(call, 0) ;
                endif
           endif
      endif

#-----------------------------------------------------
----------------------

      # add caller netline to our conference
      add_conf(call, netline, LOGON) ;

      voslog("Caller Wait for Answer Loop") ;
```

```
    # start call elapsed time
    sdate = date() ;
    stime = time() ;
    calltime = -1 ;

    for (;;)
         # compute call elapsed time
         etime = timesub(date(), time(), sdate, stime) ;

         # check for forwarded call
         if (call < FWRDMAX)
              # check for barger active
              # or forwarding subscriber answer
              if ((glb_get(BARG1SUB+subs) strneq "")
               or (glb_get(BARG2SUB+subs) strneq "")
               or (glb_get(FWRD1SUB+subs) <> 0)
               or (glb_get(FWRD2SUB+subs) <> 0))
                   # check timeouts only once per time
elapsing
                   if (etime <> calltime)
                        calltime = etime ;
                        if ((etime eq 2) or (etime eq
14))
                             # set call waiting notify
request
                             glb_set(CALLWAIT+subs,
netline) ;
                        endif
                   endif
              endif
         endif

         # check for null forwarding number
         if (glb_get(FORWARD+subs) streq "")
              # check timeouts only once per time
elapsing
              if (etime <> nulltime)
                   nulltime = etime ;
                   if (etime eq 18)
                        # leave wait for answer loop
after third ring
                        break ;
                   endif
                   if (etime eq 14)
                        # reduce apparent delay by
starting voice mail
                        # without announcement while
simulated rings
                        # are still being played
                        start_voicemail(call, 0) ;
                   endif
```

```
                    if ((etime eq 4) or (etime eq 10)
                    or (etime eq 16))
                         # play three rings

                         # add our vpline to conference
                         add_conf(call, vpline, LOGOFF) ;

                         # play ringing message
                         voslog("sc_play('ringing'
(PR0001).vox)") ;
                         sc_play(vpline, "PR0001.vox") ;

                         # delete our vpline from
conference
                         del_conf(call, vpline, netline,
LOGOFF) ;
                    endif
               endif
          endif

          # check for return answer supervision request
          if (glb_get(CALL1ANS+SUBSMAX*call+subs) <> 0)
               voslog("Answer supervison return") ;
               break ;
          endif
     endfor

     # check for null forwarding number
     # and no answer supervision return request (except
from monitor barger)
     # (negative answer supervision request indicates
set by monitor barger)
     if ((glb_get(FORWARD+subs) streq "")
     and (glb_get(CALL1ANS+SUBSMAX*call+subs) <= 0))
          # start voice mail without announcement
          # (may have already been started above)
          start_voicemail(call, 0) ;

          # add our vpline to conference
          add_conf(call, vpline, LOGOFF) ;

          # play 'one moment please' message
          voslog("sc_play('one moment'(PR0720).vox)") ;
          sc_play(vpline, "PR0720.vox") ;

          # delete our vpline from conference
          del_conf(call, vpline, netline, LOGOFF) ;
     endif

     # set wait starting time inactive
```

```
    sdate = "" ;

    voslog("Caller Conversation Loop") ;

    for (;;)
        # check for not transparent forwarding
        if (glb_get(FORWARD+subs) strneq
glb_get(CALLER_DID+subs))
            # check for barger not having answered
            # and subscriber not active on forward leg
            # and callers not in same conference
            # and forwarding inactive after having
answered
            # and voice mail not having run
            if (((call >= BARGMAX) or
(glb_get(BARG1ANS+SUBSMAX*call+subs) eq 0))
            and ((call >= FWRDMAX) or
(glb_get(FWRD1SUB+subs) eq 0))
            and ((call >= FWRDMAX) or
(glb_get(FWRD2SUB+subs) eq 0))
            and ((call >= BARGMAX) or
(glb_get(CALL1CNF+SUBSMAX*call+subs) <>
glb_get(CALL1CNF+SUBSMAX*modulo2(call)+subs)))
            and ((call >= FWRDMAX) or
(glb_get(FWRD1ACT+SUBSMAX*call+subs) eq 0))
            and ((call >= FWRDMAX) or
(glb_get(FWRD1ANS+SUBSMAX*call+subs) <> 0))
            and ((call < MAILMAX) and
(glb_get(MAIL1ACT+SUBSMAX*call+subs) streq "")))
                if (sdate streq "")
                    # save the time when this
condition begins
                    sdate = date() ;
                    stime = time() ;
                else
                    # compute condition existed time
in even secs for compares
                    etime = timesub(date(), time(),
sdate, stime)/2*2 ;
                    voslog("Elapsed waiting time
:"&etime) ;

                    # check timeouts only once per
time elapsing
                    if (etime <> calltime)
                        calltime = etime ;
                        # after first 10 seconds
                        # and last possible check for
no barger active
                        if ((etime eq 10)
                        and ((call >= BARGMAX) or
```

```
(glb_get(BARG1SUB+SUBSMAX*call+subs) eq 0)))
                                # delete our netline
from conference
                                del_conf(call, netline,
vpline, LOGOFF) ;

                                # play 'please stand by'
message
                                voslog("sc_play('please
stand by'(032).vox)") ;
                                sc_play(vpline,
"PR0032.vox") ;

                                # add our netline to
conference
                                add_conf(call, netline,
LOGOFF) ;
                            endif
                            # after first 10 secs and
every 30 secs thereafter
                            # and last possible check for
no barger active
                            if (((etime - etime/30*30) eq
10)
                            and ((call >= BARGMAX) or
(glb_get(BARG1SUB+SUBSMAX*call+subs) eq 0)))
                                # delete our netline
from conference
                                del_conf(call, netline,
vpline, LOGOFF) ;

                                # play 'press *9 to
connect to message center' message
                                voslog("sc_play('to
leave message'(687).vox)") ;
                                sc_play(vpline,
"PR0687.vox") ;

                                # add our netline to
conference
                                add_conf(call, netline,
LOGOFF) ;
                            endif
                        endif
                    endif
                endif

                # check for caller tone input
                code = sc_getdigits(vpline, 2, 3, 2) ;
                if (code eq T_MAXDT)
                    # check for forced voice mail request
```

```
                    if (sc_digits(vpline) streq "*9")
                         # check for forwarding already
active
                         if ((call < FWRDMAX)
                         and (glb_get(FWRD1SUB+subs) eq 0)
                         and (glb_get(FWRD2SUB+subs) eq
0))
                              stop_forwarding(call, 0) ;
                         endif
                         # start voice mail with
announcement
                         start_voicemail(call, 1) ;
                    endif
               endif

               # clear digits buffer
               sc_clrdigits(vpline) ;
          endif

          # check for barger inactive
          # and forwarding inactive
          # and barger having answered
          # and voice mail having run
          if   (((call >= BARGMAX) or
(glb_get(BARG1SUB+subs) streq ""))
          and  ((call >= BARGMAX) or
(glb_get(BARG2SUB+subs) streq ""))
          and  ((call >= FWRDMAX) or
(glb_get(FWRD1ACT+subs) eq 0))
          and  ((call >= FWRDMAX) or
(glb_get(FWRD2ACT+subs) eq 0))
          and  ((call >= BARGMAX) or
(glb_get(BARG1ANS+SUBSMAX*call+subs) <> 0))
          and  ((call >= MAILMAX) or
(glb_get(MAIL1ACT+SUBSMAX*call+subs) streq 0)))
               hangup() ;
               chain("GETDID", vpline) ;
          endif
     endfor
end

onsignal
     voslog("Onsignal") ;
     hangup() ;
     chain("GETDID", vpline) ;
end

func hangup()
     # abort our multitasking calls in progress
     sc_abort(vpline) ;
```

```
     # wait for all our multitasking calls to finish
     for (code = 1; (sc_stat(vpline) <> 0) and (code <=
20); code++)
          sleep(5) ;
     endfor
     if (sc_stat(vpline) <> 0)
          voslog("ERROR: 10 seconds on vpline "&vpline&"
without idle status") ;
     endif

     # put vpline onhook
     while ((code = sc_onhook(vpline)) <> T_ONH)
          voslog("ERROR: code "&code&" while putting
vpline "&vpline&" onhook") ;
          sleep(10) ;
     endwhile
     voslog("sc_onhook("&vpline&")") ;

     # stop caller's forwarding
     stop_forwarding(call, 0) ;

     # stop caller's voice mail
     stop_voicemail(call) ;

     # delete vpline from our conference
     del_conf(call, vpline, netline, LOGOFF) ;

     # delete netline from our conference
     del_conf(call, netline, vpline, LOGON) ;

     # check for active call
     if (call strneq "")
          # set our conference inactive (but leave number
usable)
          glb_set(CALL1CNF+SUBSMAX*call+subs, "+"&conf) ;
     endif

     # remove vpline receive from netline transmit
     if ((code = Sbrk(vpline, netline, SSAS_ONHOOK,
SSAL_NO)) <> 0)
          voslog("Sbrk("&vpline&","&netline&") code
"&code&" error") ;
     else
          voslog("Sbrk("&vpline&","&netline&"):
"&Sstatus(vpline)) ;
     endif

     # remove netline receive from vpline transmit
     # and enable netline SS96 alerting
     if ((code = Sbrk(netline, vpline, SSAS_ONHOOK,
SSAL_YES)) <> 0)
```

```
        voslog("Sbrk("&netline&","&vpline&") code
"&code&" error") ;
    else
        voslog("Sbrk("&netline&","&vpline&"):
"&Sstatus(netline)) ;
    endif

    # release netline/vpline allocations
    dealloc() ;

    #check for active call
    if (call strneq "")
        # set caller netline inactive
        glb_set(CALL1NET+SUBSMAX*call+subs, "") ;

        # set answer supervision inactive
        glb_set(CALL1ANS+SUBSMAX*call+subs, "") ;

        # set caller inactive (do last)
        glb_set(CALL1ACT+SUBSMAX*call+subs, "") ;
    endif

    # release our conference if unused
    rel_conf(call) ;

    # set task debug screen status
    scr_stat("Idle") ;
endfunc

func dealloc()

    # release netlinevpline allocations
    msg_put(mainpid, "D"&netline&vpline) ;
    if ((code = msg_get(3)) strneq "D-OK")
        voslog("ERROR: code "&code&" when deallocating
netline "&netline) ;
    endif
endfunc

func check_entry()

    # check for any DTMF
    if ((code = sc_getdigits(vpline, 1, 1, 1)) eq
T_MAXDT)
        # check for direct to voicemail request
        pin_entry = sc_digits(vpline);
        voslog("Caller entry: "&pin_entry) ;
        if (pin_entry streq "#")
            # play 'one moment please' message
            voslog("sc_play('one
moment'(PR0720).vox)") ;
```

```
                sc_play(vpline, "PR0720.vox") ;

                # indicate direct voicemail transfer
request
                return 1 ;
           endif
           if (pin_entry streq "0")
                # indicate short cut messages request
                return 2 ;
           endif
           # use '*' in place of null barge in code
           if ((code = glb_get(BARGE_PIN+subs)) streq "")
                code = "*" ;
           endif
           goto pin_check;

           while (sc_getdigits(vpline, 1, 3, 3) <> T_TIME)
                # add digit to PIN entry string
                pin_entry = pin_entry&sc_digits(vpline) ;
pin_check:
                if ((pin_entry streq code)
                or  (pin_entry streq
glb_get(PRGRM_PIN+subs)))
                     # request conference deallocation
                     msg_put(mainpid,
"E"&glb_get(CALL1CNF+SUBSMAX*call+subs)) ;
                     if ((code = msg_get(3)) strneq "E-OK")
                          voslog("ERROR: code "&code&" when
deallocating conference
"&glb_get(CALL1CNF+SUBSMAX*call+subs)) ;
                     else
                          voslog("Deallocating conference
"&glb_get(CALL1CNF+SUBSMAX*call+subs)) ;
                     endif
                     # set conference inactive
                     glb_set(CALL1CNF+SUBSMAX*call+subs,
"") ;

                     # set caller netline inactive
                     glb_set(CALL1NET+SUBSMAX*call+subs,
"") ;

                     # set answer supervision inactive
                     glb_set(CALL1ANS+SUBSMAX*call+subs,
"") ;

                     # set caller inactive (do last)
                     glb_set(CALL1ACT+SUBSMAX*call+subs,
"") ;

                     # send netline number to BARGER
```

```
                    msg_put(getpid(), netline) ;
                    # send subscriber number to BARGER
                    msg_put(getpid(), subs) ;
                    # send pin entry string to BARGER
                    msg_put(getpid(), pin_entry) ;
                    # replace ourself with BARGER
                    chain("BARGER", vpline) ;
               endif
          endwhile
     endif
     return 0;
endfunc

func play_reorder(scall)

     # delete our netline from caller's conference
     del_conf(scall, netline, vpline, LOGOFF) ;

     voslog("sc_playtone(reorder)") ;
     # play reorder tones for 20 seconds
     for (code = 1; code <= 40; code++)
          sc_playtone(vpline, 480, 620, -24, -24, 30) ;
          # delay between tones
          sleep(2) ;
     endfor

     # add our netline to caller's conference
     add_conf(scall, netline, LOGOFF) ;
endfunc

func start_paging();
     # check for null paging number
     if (glb_get(PAGER+subs) strneq "")
          # request paging vpline task be started
          msg_put(mainpid, "P"&subs) ;
          code = msg_get(3) ;
          if (substr(code, 1, 1) strneq "P")
               voslog("ERROR: code "&code&" when
allocating paging vpline") ;
          endif
     endif
endfunc

func start_forwarding(scall)

     if (scall < FWRDMAX)
          voslog("start_forwarding()") ;

          # request forwarding vpline task be started
          msg_put(mainpid, "F"&scall&subs) ;
          code = msg_get(3) ;
```

```
          if (substr(code, 1, 1) streq "F")
               # set forwarding task active
               glb_set(FWRD1ACT+SUBSMAX*scall+subs,
substr(code, 2, 99)) ;
          else
               voslog("ERROR: code "&code&" when
allocating forwarding vpline") ;

               # delete our netline from conference
               del_conf(scall, netline, vpline, LOGOFF) ;

               play_reorder(scall) ;

               # add our netline to conference
               add_conf(scall, netline, LOGOFF) ;

               hangup() ;
               chain("GETDID", vpline) ;
          endif
     endif
endfunc

func stop_forwarding(scall, sforce)

     # check for not forced stop request
     # and for forwarding answered
     # and for not "transparent" forwarding mode
     if ((sforce eq 0)
     and (glb_get(FWRD1ANS+SUBSMAX*scall+subs) <> 0)
     and (glb_get(FORWARD+subs) strneq
glb_get(CALLER_DID+subs)))
          return;
     endif
     sem_set(SEM_FWRD+subs) ;

     # check for forwarding active
     if (glb_get(FWRD1ACT+SUBSMAX*scall+subs) <> 0)
          voslog("stop_forwarding()") ;

          # delete forwarding netline from caller's
conference
          del_conf(scall,
glb_get(FWRD1NET+SUBSMAX*scall+subs),
glb_get(FWRD1ACT+SUBSMAX*scall+subs), LOGON) ;

          # set call answered so forward task marks
itself inactive and stopped
          glb_set(FWRD1ANS+SUBSMAX*call+subs, vpline) ;

          # abort forwarding task
          sc_hangup(glb_get(FWRD1ACT+SUBSMAX*scall+subs))
```

```
;
        voslog("sc_hangup("&glb_get(FWRD1ACT+SUBSM
AX*scall+subs)&")") ;

        # set forwarding task inactive and stopped
        glb_set(FWRD1ACT+SUBSMAX*scall+subs, 0) ;
    endif
    sem_clear(SEM_FWRD+subs) ;
endfunc

func start_voicemail(scall, smsg)

    # check for voice mail active
    if (glb_get(MAIL1ACT+SUBSMAX*scall+subs) eq 0)
        voslog("start_voicemail()") ;

        # request voice mail vpline task be started
        msg_put(mainpid, "V"&scall&subs) ;
        code = msg_get(3) ;
        if (substr(code, 1, 1) strneq "V")
            voslog("ERROR: code "&code&" when
 allocating voice mail vpline") ;

            # delete our netline from conference
            del_conf(scall, netline, vpline, LOGOFF) ;

            play_reorder(scall) ;

            # add our netline to conference
            add_conf(scall, netline, LOGOFF) ;

            hangup() ;
            chain("GETDID", vpline) ;
        else
            # set voice mail task active
            glb_set(MAIL1ACT+SUBSMAX*scall+subs,
 substr(code, 2, 99)) ;

            if (smsg <> 0)
                # delete our netline from conference
                del_conf(scall, netline, vpline,
 LOGOFF) ;

                # play 'connecting call to voice mail'
 message
                voslog("sc_play('to voice
 mail'(030).vox)") ;
                sc_play(vpline, "PR0030.vox") ;

                # add our netline to conference
                add_conf(scall, netline, LOGOFF) ;
```

```
            endif
        endif
    endif
endfunc

func stop_voicemail(scall)

    sem_set(SEM_MAIL+subs) ;

    # check for voice mail active
    if (glb_get(MAIL1ACT+SUBSMAX*scall+subs) <> 0)
        voslog("stop_voicemail()") ;

        # delete voice mail netline from caller's
conference
        del_conf(scall,
glb_get(MAIL1NET+SUBSMAX*scall+subs),
glb_get(MAIL1ACT+SUBSMAX*scall+subs), LOGON) ;

        # abort voice mail task
        sc_hangup(glb_get(MAIL1ACT+SUBSMAX*scall+subs))
;
        voslog("sc_hangup("&glb_get(MAIL1ACT+SUBSM
AX*scall+subs)&")") ;

        # set voice mail task inactive and stopped
        glb_set(MAIL1ACT+SUBSMAX*scall+subs, 0) ;
    endif
    sem_clear(SEM_MAIL+subs) ;
endfunc

func modulo2(scall)
    return   ((scall+1)-(scall+1)/2*2) ;
endfunc

    include "confsubs.inc"      # conferencing
subroutines

-------------------------------------------------------------
--------

#
#    CALLOUT.VS - D121 line task
#
#    Designed to work in concert with GETDID.VS
#
#-------------------------------------------------------------
-----------------------

dec
    include "datmod.inc"        # SS96 library
```

```
definitions
    include "global.inc"          # define global variable
numbers
    var code:16 ;                 # return value from function
calls
    var vpline:3 ;                # voice processing line
number
    var netline:3 ;               # network line number
    var conf:3 ;                  # conference number
    var subs:3 ;                  # subscriber number
    var call:3 ;                  # call number
    var mainpid:3 ;               # main task PID

    var stats:16 ;                # beginning netline
status (confsubs)
    var line:3 ;                  # needed for scr_stat
end

program

# Initializations
    mainpid = glb_get(MAINPID) ;

    vpline = arg() ;
    if ((vpline < GDIDBGN) or (vpline > GDIDEND))
        voslog("ERROR: invalid vpline number "&vpline&"
message received") ;
        hangup() ;
        chain("GETDID", vpline) ;
    endif
    line = vpline ;                          # debug only

# Begin call processing
    # enable hang up jump to onsignal
    sc_use(vpline) ;

    # set call number inactive (in case of unexpected
disconnect)
    call = "" ;

    # do not allow disconnects to run onsignal
    sc_watch(vpline, "+-", 1) ;

    while (length(netline = msg_get(3)) eq 0)
    endwhile
    if ((netline < NETBGN) or (netline > NETEND))
        if ((netline < ALTBGN) or (netline > ALTEND))
            voslog("ERROR: invalid netline number
"&netline&" message received") ;
            hangup() ;
```

```
            chain("GETDID", vpline) ;
        endif
    endif

    while (length(subs = msg_get(3)) eq 0)
    endwhile
    if ((subs < 0) or (subs >= SUBSMAX))
        voslog("ERROR: invalid subscriber number
"&subs&" message received") ;
        hangup() ;
        chain("GETDID", vpline) ;
    endif

    # set task debug screen status
    scr_stat("C-"&netline) ;

    while (length(call = msg_get(3)) eq 0)
    endwhile
    if ((call < 0) or (call >= BARGMAX))
        voslog("ERROR: invalid outbound call number
"&call&" message received") ;
        hangup() ;
        chain("GETDID", vpline) ;
    endif

    # return answer supervision via vpline
    while ((code = sc_offhook(vpline)) <> T_OFFH)
        voslog("ERROR: code "&code&" while taking
vpline "&vpline&" offhook") ;
        sleep(10) ;
    endwhile

    # now allow disconnects to run onsignal
    sc_watch(vpline, "d+-", 1) ;

    # set task debug status
    scr_stat("O-"&netline) ;

    for (;;)
        # clear digits buffer
        sc_clrdigits(vpline) ;

        # check for caller tone input
        code = sc_getdigits(vpline, 2, 3, 2) ;
        if (code eq T_MAXDT)
            # check for forced voice mail request
            if (sc_digits(vpline) streq "*9")
                # start voice mail with announcement
                start_voicemail(call, 1) ;
            endif
        endif
```

```
        # check for barger not active
        if ((glb_get(BARG1SUB+subs) eq 0)
        and (glb_get(BARG2SUB+subs) eq 0))
             hangup() ;
             chain("GETDID", vpline) ;
        endif
    endfor
end

onsignal
    voslog("Onsignal") ;
    hangup() ;
    chain("GETDID", vpline) ;
end

func hangup()
    # abort our multitasking calls in progress
    sc_abort(vpline) ;

    # wait for all our multitasking calls to finish
    for (code = 1; (sc_stat(vpline) <> 0) and (code <=
20); code++)
         sleep(5) ;
    endfor
    if (sc_stat(vpline) <> 0)
         voslog("ERROR: 10 seconds on vpline "&vpline&"
without idle status") ;
    endif

    # put vpline onhook
    while ((code = sc_onhook(vpline)) <> T_ONH)
         voslog("ERROR: code "&code&" while putting
vpline "&vpline&" onhook") ;
         sleep(10) ;
    endwhile
    voslog("sc_onhook("&vpline&")") ;

    # delete vpline from our conference
    del_conf(call, vpline, netline, LOGOFF) ;

    # delete netline from our conference
    del_conf(call, netline, vpline, LOGON) ;

    # check for active call
    if (call strneq "")
         # set our conference inactive (but leave number
usable)
         glb_set(CALL1CNF+SUBSMAX*call+subs, "+"&conf) ;
    endif
```

```
    # remove vpline receive from netline transmit
    if ((code = Sbrk(vpline, netline, SSAS_ONHOOK,
SSAL_NO)) <> 0)
        voslog("Sbrk("&vpline&","&netline&") code
"&code&" error") ;
    else
        voslog("Sbrk("&vpline&","&netline&"):
"&Sstatus(vpline)) ;
    endif

    # remove netline receive from vpline transmit
    # and enable netline SS96 alerting
    if ((code = Sbrk(netline, vpline, SSAS_ONHOOK,
SSAL_YES)) <> 0)
        voslog("Sbrk("&netline&","&vpline&") code
"&code&" error") ;
    else
        voslog("Sbrk("&netline&","&vpline&"):
"&Sstatus(netline)) ;
    endif

    # release netline/vpline allocations
    dealloc() ;

    #check for active call
    if (call strneq "")
        # set caller netline inactive
        glb_set(CALL1NET+SUBSMAX*call+subs, "") ;

        # set answer supervision inactive
        glb_set(CALL1ANS+SUBSMAX*call+subs, "") ;

        # set caller inactive (do last)
        glb_set(CALL1ACT+SUBSMAX*call+subs, "") ;
    endif

    # release our conference if unused
    rel_conf(call) ;

    # set task debug screen status
    scr_stat("Idle") ;
endfunc

func dealloc()

    # release netlinevpline allocations
    msg_put(mainpid, "D"&netline&vpline) ;
    if ((code = msg_get(3)) strneq "D-OK")
        voslog("ERROR: code "&code&" when deallocating
netline "&netline) ;
    endif
```

```
endfunc

func start_voicemail(scall, smsg)

     # check for voice mail active
     if (glb_get(MAIL1ACT+SUBSMAX*scall+subs) eq 0)
          # request voice mail vpline task be started
          msg_put(mainpid, "V"&scall&subs) ;
          code = msg_get(3) ;
          if (substr(code, 1, 1) strneq "V")
               voslog("ERROR: code "&code&" when
allocating voice mail vpline") ;

               # delete our netline from conference
               del_conf(scall, netline, vpline, LOGOFF) ;

               play_reorder(scall) ;

               # add our netline to conference
               add_conf(scall, netline, LOGOFF) ;

               hangup() ;
               chain("GETDID", vpline) ;
          else
               # set voice mail task active
               glb_set(MAIL1ACT+SUBSMAX*scall+subs,
substr(code, 2, 99)) ;

               if (smsg <> 0)
                    # delete our netline from conference
                    del_conf(scall, netline, vpline,
LOGOFF) ;

                    # play 'connecting call to voice mail'
message
                    voslog("sc_play('to voice
mail'(030).vox)") ;
                    sc_play(vpline, "PR0030.vox") ;

                    # add our netline to conference
                    add_conf(scall, netline, LOGOFF) ;
               endif
          endif
     endif
endfunc

func play_reorder(scall)

     # delete our netline from caller's conference
     del_conf(scall, netline, vpline, LOGOFF) ;
```

```
    voslog("sc_playtone(reorder)") ;
    # play reorder tones for 20 seconds
    for (code = 1; code <= 40; code++)
        sc_playtone(vpline, 480, 620, -24, -24, 30) ;
        # delay between tones
        sleep(2) ;
    endfor

    # add our netline to caller's conference
    add_conf(scall, netline, LOGOFF) ;
endfunc

func modulo2(scall)
    return   ((scall+1)-(scall+1)/2*2) ;
endfunc

    include "confsubs.inc"      # conferencing
subroutines
```

```
#
#    GETDID.VS - D121 line task
#
#    Designed to work in concert with MAIN.VS
#
#-------------------------------------------------------
------------------------

dec
    include "datmod.inc"        # SS96 library
definitions
    include "global.inc"        # define global variable
numbers

    var code:16 ;           # return value from function
calls
    var cnt:3 ;             # general counter
    var vpline:3 ;              # voice processing line
number
    var netline:3 ;             # network line number
    var dir:4 ;             # call direction ("In"/"Out")
    var subs:3 ;            # subscriber number
    var call:3 ;            # call number
    var mainpid:3 ;             # main task PID
    var did:10 ;            # DID

    var line:3 ;            # needed for scr_stat
end
```

```
program

# Initializations
     vpline = arg() ;
     line = vpline ;                    # debug only
     if (glb_get(MAINFLG) <> 99)             # if first time
initialization
          mainpid = glb_get(MAINPID) ;
          msg_put(mainpid, getpid()) ;    # inform main
task of our pid

          # wait for main task to be ready to go
          while (glb_get(MAINFLG) <> 99)
          endwhile
     endif

# Begin call processing
     # enable jump to onsignal
     sc_use(vpline) ;

# Set minimum detected wink duration to 50 msec (from
100ms default)
     sc_setparm(vpline, 518, 5) ;

# Set maximum detected wink duration to 300 msec (from
200ms default)
     sc_setparm(vpline, 519, 30) ;

# Turn on speech card function calls trace
#    sc_trace(vpline, 1) ;

# Attempt to go onhook
     # do not allow disconnects to run onsignal
     sc_watch(vpline, "+-", 1);

     while ((code = sc_onhook(vpline)) <> T_ONH)
          voslog("ERROR: "&code&" - onhook failure");
          sleep(10) ;
     endwhile

# Wait for message from main task indicating start of
call
     for (;;)
          while (length(vpline = msg_get(3600)) eq 0)
          endwhile
          if ((vpline < GDIDBGN) or (vpline > GDIDEND))
               voslog("ERROR: invalid vpline number
"&vpline&" message received") ;
               continue ;
          endif
```

```
          while (length(netline = msg_get(3)) eq 0)
          endwhile
          if ((netline < NETBGN) or (netline > NETEND))
               if ((netline < ALTBGN) or (netline >
ALTEND))
                    voslog("ERROR: invalid netline number
"&netline&" message received") ;
                    continue ;
               endif
          endif

          while (length(dir = msg_get(3)) eq 0)
          endwhile
          if ((dir strneq "In") and (dir strneq "Out"))
               voslog("ERROR: invalid call direction
"&dir&" message received") ;
               continue ;
          endif

          # set task debug screen status
          scr_stat("G-"&netline) ;

          # defer hang up jump to onsignal
          sc_sigctl("d") ;

          # drive vpline receive from netline transmit
          if ((code = Scon(vpline, netline, SSAS_PASS,
SSAL_NO)) <> 0)
               voslog("Scon("&vpline&","&netline&") code
"&code&" error") ;
          else
               voslog("Scon("&vpline&","&netline&"):
"&Sstatus(vpline));
          endif

          # check for inbound call direction
          if (dir streq "In")
               # drive netline receive from vpline
transmit
               if ((code = Scon(netline, vpline,
SSAS_PASS, SSAL_YES)) <> 0)
                    voslog("Scon("&netline&","&vpline&")
code "&code&" error") ;
               else
                    voslog("Scon("&netline&","&vpline&"):
"&Sstatus(netline));
               endif
          endif

          # throw away deferred hang ups
          sc_sigctl("c") ;
```

```
        # check for outbound call direction
        if (dir streq "Out")
             while (length(subs = msg_get(3)) eq 0)
             endwhile
             if ((subs < 0) or (subs > SUBSMAX))
                  voslog("ERROR: invalid subscriber
number "&subs&" message received") ;
                  continue ;
             endif

             while (length(call = msg_get(3)) eq 0)
             endwhile
             if ((call < 0) or (call > CALLMAX))
                  voslog("ERROR: invalid call number
"&call&" message received") ;
                  continue ;
             endif

             # chain to CALLOUT if any calls inactive
             if ((glb_get(CALL1ACT+subs) eq 0)
              or (glb_get(CALL2ACT+subs) eq 0)
              or (glb_get(CALL3ACT+subs) eq 0)
              or (glb_get(CALL4ACT+subs) eq 0))
                  # send netline number to CALLOUT
                  msg_put(getpid(), netline) ;
                  # send subscriber number to CALLOUT
                  msg_put(getpid(), subs) ;
                  # send call number to CALLOUT
                  msg_put(getpid(), call) ;
                  # replace ourself with CALLOUT
                  chain("CALLOUT", vpline) ;
             else
                  # call exceeds our capacity
                  voslog("sc_playtone(busy)") ;
                  # play busy tones for 20 seconds
                  for (cnt = 1; cnt <= 20; cnt++)
                       sc_playtone(vpline, 480, 620, -
 24, -24, 50) ;

                       # delay between tones
                       sleep(5) ;
                  endfor
                  hangup() ;
                  # get next call to process
                  continue ;
             endif
        endif

        # set up for later transition checks
        sc_clrtrans(vpline) ;
```

```
          # disable tone interruption
          sc_toneint(vpline, 0) ;

          # defer hang up jump to onsignal
          sc_sigctl("(") ;

          # send wink to get DID
          while ((code = sc_wink(vpline)) <> T_WINK)
               voslog("ERROR: code "&code&" while sending
wink on vpline "&vpline) ;
               sleep(10) ;
          endwhile

          # set MF detection parameter
#         MF_setparm(vpline, 1, 0) ;      # MF

          # switch to MF detection
#         sc_digitctl(vpline, 0, 1) ;          # MF
#         MF_on(vpline) ;                 # MF

          # clear digit buffer
          sc_clrdigits(vpline) ;

          # WATCH DIGIT COUNT !!!
          if ((code = sc_getdigits(vpline, 7, 4, 1)) <>
T_MAXDT) # DTMF
#         if ((code = sc_getdigits(vpline, 9, 4, 1)) <>
T_MAXDT) # MF
               voslog("ERROR: "&code&" - accessing DID
failure");
               voslog("Incomplete DID: "&did&":");
               hangup() ;
               continue ;
          endif
          did = sc_digits(vpline) ;

          voslog("Inbound DID: "&did) ;

          # switch back to DTMF detection
#         MF_off(vpline) ;            # MF
#         sc_digitctl(vpline, 1, 1) ;         # MF

          # allow deferred hang ups
          sc_sigctl(")") ;

          # enable tone interruption
          sc_toneint(vpline, 1) ;

          # search global tables indexed by subscribers
          for (subs = 0; subs < SUBSMAX; subs++)
```

```
            # check if processing a caller access
number call
            if (did streq glb_get(CALLER_DID+subs))
                # chain to CALLIN if any calls
inactive
                if ((glb_get(CALL1ACT+subs) eq 0)
                 or (glb_get(CALL2ACT+subs) eq 0)
                 or (glb_get(CALL3ACT+subs) eq 0)
                 or (glb_get(CALL4ACT+subs) eq 0))
                    # send netline number to CALLIN
                    msg_put(getpid(), netline) ;
                    # send subscriber number to
CALLIN
                    msg_put(getpid(), subs) ;
                    # replace ourself with CALLIN
                    chain("CALLIN", vpline) ;
                else
                    break ;
                endif
            endif

            # check if processing a barger access
number call
            if (did streq glb_get(BARGE_DID+subs))
                # chain to BARGER if barger inactive
                if ((glb_get(BARG1SUB+subs) streq "")
                and (glb_get(BARG2SUB+subs) streq ""))
                    # send netline number to BARGER
                    msg_put(getpid(), netline) ;
                    # send subscriber number to
BARGER
                    msg_put(getpid(), subs) ;
                    # send initial pin entry string
to BARGER
                    msg_put(getpid(), " ") ;
                    # replace ourself with BARGER
                    chain("BARGER", vpline) ;
                else
                    break ;
                endif
            endif
        endfor
        if (subs < SUBSMAX)
            # DID found but call exceeds our capacity
            voslog("sc_playtone(busy)") ;
            # play busy tones for 20 seconds
            for (cnt = 1; cnt <= 20; cnt++)
                sc_playtone(vpline, 480, 620, -24, -
24, 50) ;
                # delay between tones
```

```
                sleep(5) ;
            endfor
        else
            voslog("ERROR: no match found for DID:
"&did);
            # play reorder tones for 20 seconds
            voslog("sc_playtone(reorder)") ;
            for (cnt = 1; cnt <= 40; cnt++)
                sc_playtone(vpline, 480, 620, -24, -
24, 30) ;

                # delay between tones
                sleep(2) ;
            endfor
        endif
        hangup() ;
    endfor
end

#-------------------------------------------------------
----------------------

# no onsignal needed
#onsignal
#    voslog("Onsignal") ;
#    hangup() ;
#    restart ;
#end

func hangup()

    # abort our multitasking calls in progress
    sc_abort(vpline) ;

    # wait for all our multitasking calls to finish
    for (code = 1; (sc_stat(vpline) <> 0) and (code <=
20); code++)
        sleep(5) ;
    endfor
    if (sc_stat(vpline) <> 0)
        voslog("ERROR: 10 seconds on vpline "&vpline&"
without idle status") ;
    endif

    # put vpline onhook
    while ((code = sc_onhook(vpline)) <> T_ONH)
        voslog("ERROR: code "&code&" while putting
vpline "&vpline&" onhook") ;
        sleep(10) ;
    endwhile
    voslog("sc_onhook("&vpline&")") ;
```

```
    # remove vpline receive from netline transmit
    if ((code = Sbrk(vpline, netline, SSAS_ONHOOK,
SSAL_NO)) <> 0)
        voslog("Sbrk("&vpline&","&netline&") code
"&code&" error") ;
    else
        voslog("Sbrk("&vpline&","&netline&"):
"&Sstatus(vpline));
    endif

    # remove netline receive from vpline transmit
    # and enable netline SS96 alerting
    if ((code = Sbrk(netline, vpline, SSAS_ONHOOK,
SSAL_YES)) <> 0)
        voslog("Sbrk("&netline&","&vpline&") code
"&code&" error") ;
    else
        voslog("Sbrk("&netline&","&vpline&"):
"&Sstatus(netline));
    endif

    # release netline/vpline allocations
    msg_put(mainpid, "D"&netline&vpline) ;
    if ((code = msg_get(3)) strneq "D-OK")
        voslog("ERROR: code "&code&" when deallocating
netline "&netline) ;
    endif

    # set task debug screen status
    scr_stat("Idle") ;
endfunc
```

APPENDIX III

SOURCE CODE LISTING OF A FORWARD TASK
USED BY THE MAIN PROGRAM

```
#
#    FORWARD.VS - D121 line task
#
#    Designed to work in concert with MAIN.VS
#
#-------------------------------------------------------------
-----------------------

dec
    include "datmod.inc"          # SS96 library
definitions
    include "global.inc"          # define global variable
numbers

    var code:16 ;              # return value from function
calls
    var vpline:3 ;
    var netline:3 ;
    var mainpid:3 ;
    var subs:3 ;               # subscriber number
    var call:3 ;               # toggling call number
    var conf:3 ;               # conference number
    var flash:3 ;              # flash DTMF digits
    var car:3 ;                # call analysis result
    var sdate:7 ;              # starting date
    var stime:7 ;              # starting time
    var fwrdnum: 20 ;          # dialed forwarding number

    var stats:16 ;                 # beginning netline
status (confsubs)
    var line:3 ;               # needed for scr_stat
end

program

# Initializations
    vpline = arg() ;
    line = vpline ;                    # debug only
    if (glb_get(MAINFLG) <> 99)        # if first time
initialization
        mainpid = glb_get(MAINPID) ;

        # inform main task of our pid
        msg_put(mainpid, getpid()) ;
```

```
        # wait for main task to be ready to go
        while (glb_get(MAINFLG) <> 99)
        endwhile
    endif

# Begin call processing
    sc_use(vpline) ;

# Set minimum detected wink duration to 70 msec (from
100ms default)
    sc_setparm(vpline, 518, 7) ;

# Set maximum detected wink duration to 250 msec (from
200ms default)
    sc_setparm(vpline, 519, 25) ;

# Turn on speech card function calls trace
#   sc_trace(vpline, 1) ;

# Wait for onhook complete
    sc_watch(vpline, "+-", 1) ;
    while ((code = sc_onhook(vpline)) <> T_ONH)
        voslog("ERROR: code "&code&" - onhook failure")
;
        sleep(10) ;
    endwhile

# Wait for message from main task indicating start of
call
    for (;;)
        # initialize netline in case of reorder
        netline = 0 ;

        # ignore disconnect
        sc_watch(vpline, "lw+-", 1) ;

        while (length(vpline = msg_get(3600)) eq 0)
        endwhile
        if ((vpline < FWRDBGN) or (vpline > FWRDEND))
            voslog("ERROR: invalid vpline number
"&vpline&" message received") ;
            # set forwarding inactive
# (call not set
yet) glb_set(FWRD1ACT+SUBSMAX*call+subs, "") ;
            continue ;
        endif

        # get subscriber number
        while (length(subs = msg_get(3)) eq 0)
        endwhile
```

```
          if ((subs < 0) or (subs >= SUBSMAX))
               voslog("ERROR: invalid subscriber number
"&subs&" message received") ;
               # set forwarding inactive
# (call not set
yet) glb_set(FWRD1ACT+SUBSMAX*call+subs, "") ;
               continue ;
          endif

          # get call number
          while (length(call = msg_get(3)) eq 0)
          endwhile
          if ((call < 0) or (call >= FWRDMAX))
               voslog("ERROR: invalid call number
"&call&" message received") ;
               # set forwarding inactive
# (call #
invalid) glb_set(FWRD1ACT+SUBSMAX*call+subs, "") ;
               continue ;
          endif

          # indicate dialing forwarding number
          glb_set(FWRD1ANS+SUBSMAX*call+subs, 0) ;

          # check for "transparent" forwarding mode
          if (glb_get(FORWARD+subs) streq
glb_get(CALLER_DID+subs))
               # request outbound alternate netline
allocation
               msg_put(mainpid, "A"&vpline) ;
               if ((code = msg_get(3)) strneq "A-OK")
                    voslog("ERROR: code "&code&" when
allocating outbound netline") ;
                    # set forwarding inactive
                    glb_set(FWRD1ACT+SUBSMAX*call+subs,
"") ;
                    # set forwarding call unanswered
                    glb_set(FWRD1ANS+SUBSMAX*call+subs,
"") ;
                    # start voicemail without announcement
                    start_voicemail(call, 0) ;
                    continue ;
               endif

               # get allocated alternate netline
               while (length(netline = msg_get(3)) eq 0)
               endwhile
               if ((netline < ALTBGN) or (netline >
ALTEND))
                    voslog("ERROR: invalid alternate
netline number "&netline&" message") ;
```

```
                    # set forwarding inactive
                    glb_set(FWRD1ACT+SUBSMAX*call+subs,
"") ;
                    # set forwarding call unanswered
                    glb_set(FWRD1ANS+SUBSMAX*call+subs,
"") ;
                    # release netline/vpline allocations
                    dealloc() ;
                    continue ;
               endif
          else
               # request outbound netline allocation
               msg_put(mainpid, "N"&vpline) ;
               if ((code = msg_get(3)) strneq "N-OK")
                    voslog("ERROR: code "&code&" when
allocating outbound netline") ;
                    # set forwarding inactive
                    glb_set(FWRD1ACT+SUBSMAX*call+subs,
"") ;
                    # set forwarding call unanswered
                    glb_set(FWRD1ANS+SUBSMAX*call+subs,
"") ;
                    # start voicemail without announcement
                    start_voicemail(call, 0) ;
                    continue ;
               endif

               # get allocated netline
               while (length(netline = msg_get(3)) eq 0)
               endwhile
               if ((netline < NETBGN) or (netline >
NETEND))
                    voslog("ERROR: invalid netline number
"&netline&" message") ;
                    # set forwarding inactive
                    glb_set(FWRD1ACT+SUBSMAX*call+subs,
"") ;
                    # set forwarding call unanswered
                    glb_set(FWRD1ANS+SUBSMAX*call+subs,
"") ;
                    # release netline/vpline allocations
                    dealloc() ;
                    continue ;
               endif
          endif

          # set our netline active
          glb_set(FWRD1NET+SUBSMAX*call+subs, netline) ;

          #defer hang up jump to onsignal
          sc_sigctl("d") ;
```

```
          # drive netline receive from vpline transmit
          if ((code = Scon(netline, vpline, SSAS_PASS,
SSAL_YES)) <> 0)
               voslog("Scon("&netline&","&vpline&") code
"&code&" error") ;
          else
               voslog("Scon("&netline&","&vpline&"):
"&Sstatus(netline)) ;
          endif

          # drive vpline receive from netline transmit
          if ((code = Scon(vpline, netline, SSAS_PASS,
SSAL_NO)) <> 0)
               voslog("Scon("&vpline&","&netline&") code
"&code&" error") ;
          else
               voslog("Scon("&vpline&","&netline&"):
"&Sstatus(vpline)) ;
          endif

          #throw away deferred hang ups
          sc_sigctl("c") ;

          # set task debug screen status
          scr_stat("F-"&netline) ;

          # save the initial time
          sdate = date() ;
          stime = time() ;
dial_fwrd:
          # go off hook
          if ((code = sc_offhook(vpline)) <> T_OFFH)
               voslog("ERROR: code "&code&" while putting
vpline "&vpline&" offhook") ;
               goto retry ;
          else
               voslog("sc_offhook("&vpline&")") ;
          endif

          # wait one second for next event report
          code = sc_wait(vpline, 10) ;
          voslog("sc_wait("&vpline&",10) = "&code) ;

          # check for loop current event report
          if (code <> T_LCON)
               voslog("ERROR: Loop current timeout on
vpline "&vpline) ;
               goto retry ;
          endif
```

```
        # wait one second for next event report
        code = sc_wait(vpline, 10) ;
        voslog("sc_wait("&vpline&",10) = "&code) ;

        # check for wink event report
        if (code <> T_WKRECV)
            voslog("ERROR: Wink timeout on vpline
"&vpline) ;
            goto retry ;
        endif

        # get full dialing string for trunk group used
        if ((netline >= NETBGN) and (netline <=
NETEND))
            fwrdnum =
glb_get(PSTN_PRE+subs)&glb_get(FORWARD+subs)&glb_get
(PSTN_PST+subs) ;
        else
            fwrdnum =
glb_get(STRATUS_PRE)&glb_get(FORWARD+subs)&glb_get
(STRATUS_PST) ;
        endif

        voslog("Dialing forwarding number "&fwrdnum) ;

        # dial the forwarding number - wait until
dialing completed
        if ((code = sc_dial(vpline, fwrdnum)) <>
T_DIAL)
            voslog("ERROR: code "&code&" while dialing
forwarding number") ;
retry:
            if (timesub(date(), time(), sdate, stime)
<= 10)
                # put vpline onhook
                if ((code = sc_onhook(vpline)) <>
T_ONH)
                    voslog("ERROR: code "&code&"
while putting vpline "&vpline&" onhook") ;
                else
                    voslog("sc_onhook("&vpline&")") ;
                endif

                # wait for onhook recognition
                sleep(10) ;

                goto dial_fwrd ;
            else
                goto call_busy ;
            endif
        endif
```

```
        # add our netline to current caller's
conference
        add_conf(call, netline, LOGON) ;

        # set the initial time
        sdate = date() ;
        stime = time() ;

        # set call progress active (MAIN task needs
netline)
        glb_set(FWRD1CPA+SUBSMAX*call+subs, netline) ;

        # start call analysis
        sc_call(vpline, ",") ;

        # set call progress inactive
        glb_set(FWRD1CPA+SUBSMAX*call+subs, "") ;

        # get call analysis result
        car = sc_getcar(vpline) ;

        switch (car)

        case CA_BUSY:     # Called line is busy
            voslog("CAR - vpline "&vpline&" is busy")
;
        case CA_NOAN:     # Called line did not answer
            voslog("CAR - vpline "&vpline&" did not
answer") ;
        case CA_NORNG:          # Called line did not
ring
            voslog("CAR - vpline "&vpline&" did not
ring") ;
        case CA_CONN:     # Called line connected
            voslog("CAR - vpline "&vpline&"
connected") ;
        case CA_OPINT:          # Called line received
operator intercept
            voslog("CAR - vpline "&vpline&" received
operator intervept") ;
        case T_LCTERM:          # MAIN task aborted CAR
due to answer supervision
            voslog("CAR aborted - vpline "&vpline&"
answer supervision") ;
        endswitch

        # start voicemail if call busy
        if (car eq CA_BUSY)
call_busy:
            # delete our netline from caller's
```

```
conference
              del_conf(call, netline, vpline, LOGOFF) ;

              # set forwarding call unanswered
              glb_set(FWRD1ANS+SUBSMAX*call+subs, "") ;

              # abort possible caller's task sc_play
              sc_abort(glb_get(CALL1ACT+SUBSMAX*cal
l+subs)) ;
              voslog("sc_abort("&glb_get(CALL1ACT
+SUBSMAX*call+subs)&")") ;

              # add our vpline to caller's conference
              add_conf(call, vpline, LOGOFF) ;

              # play 'one moment please' message
              voslog("sc_play('one
moment'(PR0720).vox)") ;
              sc_play(vpline, "PR0720.vox") ;

              # delete our vpline from conference
              del_conf(call, vpline, netline, LOGOFF) ;

              # start voice mail without announcement
              start_voicemail(call, 0) ;

              # disconnect and look for new forwarding
request
              hangup() ;
              continue ;
         endif

         # check for maximum time for audio feedback
from forwarding channel
         while (timesub(date(), time(), sdate, stime) <
25)
              # check for answer supervision
              if (sc_stat(vpline, 9) eq 0)
                   voslog("Answer supervision") ;

                   # TEST - monitor forwarding answer
function
                   if (glb_get(CALLER_DID+subs) streq
"6545786")
                        # delete our netline from
caller's conference
                        del_conf(call, netline, vpline,
LOGOFF) ;

                        # abort possible caller's task
sc_play
```

```
                        sc_abort(glb_get(CALL1ACT
+SUBSMAX*call+subs)) ;
                        voslog("sc_abort("&glb_get
(CALL1ACT+SUBSMAX*call+subs)&")") ;

                        # add our vpline to caller's
conference
                        add_conf(call, vpline, LOGOFF) ;

                        # play 'one moment please'
message
                        voslog("sc_play('one
moment'(PR0720).vox)") ;
                        sc_play(vpline, "PR0720.vox") ;

                        # delete our vpline from
conference
                        del_conf(call, vpline, netline,
LOGOFF) ;

                        # start voice mail without
announcement
                        start_voicemail(call, 0) ;
                    endif
                    break ;
                endif
            endwhile
            # check for no answer supervision
            if (sc_stat(vpline, 9) eq 1)
                # delete our netline from caller's
conference
                del_conf(call, netline, vpline, LOGOFF) ;

                # abort possible caller's task sc_play
                sc_abort(glb_get(CALL1ACT+SUBSMAX*cal
l+subs)) ;
                voslog("sc_abort("&glb_get(CALL1ACT
+SUBSMAX*call+subs)&")") ;

                # delete caller's netline from conference
                del_conf(call,
glb_get(CALL1NET+SUBSMAX*call+subs),
glb_get(CALL1ACT+SUBSMAX*call+subs), LOGON) ;

                # play 'one moment please' message to
caller's netline
                voslog("sc_play('one
moment'(PR0720).vox)") ;
                sc_play(glb_get(CALL1ACT+SUBSMAX*call
+subs), "PR0720.vox") ;
```

```
            # add caller's netline to conference
            add_conf(call,
glb_get(CALL1NET+SUBSMAX*call+subs), LOGON) ;

            # start voice mail without announcement
            start_voicemail(call, 0) ;

            # check for maximum time for forwarding
answer
            while (timesub(date(), time(), sdate,
stime) < 64)
                # check for answer supervision
                if (sc_stat(vpline, 9) eq 0)
                    voslog("Answer supervision") ;
                    break ;
                endif
                # play awaiting answer supervision
tones
                sc_playtone(vpline, 1000, 2000, -30, -
30, 100) ;
            endwhile
            # check for no answer supervision
            if (sc_stat(vpline, 9) eq 1)
                # set forwarding call unanswered
                glb_set(FWRD1ANS+SUBSMAX*call+subs,
"") ;

                # disconnect and look for new
forwarding request
                hangup() ;
                continue ;
            endif
        endif

        # TEST - monitor forwarding answer function
        if (glb_get(CALLER_DID+subs) streq "6545786")
            # now watch for disconnect (duplicated
below)
            sc_watch(vpline, "dl+-", 1) ;

            # add our netline as listener to selected
caller's conference
            add_list(call, netline, LOGON) ;

            for (;;)
                # delete our netline as listener
                del_list(call, netline, vpline,
LOGOFF) ;

                # play monitor mode beep to listener
                sc_playtone(vpline, 1000, 2000, -30, -
```

```
30, 20) ;

                    # add our netline as listener
                    add_list(call, netline, LOGOFF) ;

                    # clear digit buffer
                    sc_clrdigits(vpline) ;

                    # space monitor beeps every 5 seconds
while looking for tone input
                    if ((code = sc_getdigits(vpline, 1, 5,
1)) eq T_MAXDT)
                         # check for back out request
                         if ((code = sc_digits(vpline))
streq "#")
                              # delete our netline as
listener to selected caller's conference
                              del_list(call, netline,
vpline, LOGON) ;

                              # disconnect and look for new
forwarding request
                              hangup() ;
                              continue ;
                         endif
                         # check for proceed request
                         if (code streq "*")
                              break;
                         endif
                    endif
               endfor

               # delete our netline as listener to
current caller's conference
               del_list(call, netline, vpline, LOGON) ;

               # add our netline to current caller's
conference
               add_conf(call, netline, LOGON) ;

               # add our vpline to current caller's
conference
               add_conf(call, vpline, LOGOFF) ;

               # play joining conference tones
               sc_playtone(vpline, 1000, 2000, -30, -30,
100) ;
               voslog("sc_playtone(join conf)") ;

               # delete our vpline from current caller's
conference
```

```
            del_conf(call, vpline, netline, LOGOFF) ;
        else
            # add our netline to current caller's
conference
            add_conf(call, netline, LOGON) ;

        endif

        # stop voice mail if active
        stop_voicemail(call) ;

        # set voice mail task inactive and not run
        glb_set(MAIL1ACT+SUBSMAX*call+subs, "") ;

        # check for MAIN task stopping call progress
due to answer supervision
        if (car <> T_LCTERM)
            # abort possible caller's task sc_play
(e.g. ringing)
            sc_abort(glb_get(CALL1ACT+SUBSMAX*cal
l+subs)) ;
            voslog("sc_abort("&glb_get(CALL1ACT
+SUBSMAX*call+subs)&")") ;

            # request answer supervision be returned
to caller
            glb_set(CALL1ANS+SUBSMAX*call+subs,
vpline) ;
        endif

        # set forwarding call answered
        glb_set(FWRD1ANS+SUBSMAX*call+subs, vpline) ;

        # now watch for disconnect
        sc_watch(vpline, "dl+-", 1) ;

        # check for barging subscriber inactive
        # and forwarding subscriber inactive
        if ((glb_get(BARG1SUB+subs) streq "")
        and (glb_get(BARG2SUB+subs) streq "")
        and (glb_get(FWRD1SUB+subs) eq 0)
        and (glb_get(FWRD2SUB+subs) eq 0))
            # must assume for now the subscriber
answered

            # set our forwarding subscriber active
            glb_set(FWRD1SUB+SUBSMAX*call+subs,
netline) ;

            # reset call waiting request
            glb_set(CALLWAIT+subs, "") ;
```

```
          endif

          voslog("Forwarding Conversation Loop") ;

          for (;;)
               # check for barging subscriber active
               if ((glb_get(BARG1SUB+subs) strneq "")
               or  (glb_get(BARG2SUB+subs) strneq ""))
                    # set both forwarding subscribers
inactive
                    glb_set(FWRD1SUB+subs, "") ;
                    glb_set(FWRD2SUB+subs, "") ;
               endif

               # check for subscriber active on our
netline
               if ((glb_get(FWRD1SUB+subs) eq netline)
                or (glb_get(FWRD2SUB+subs) eq netline))
                    # check for call waiting request
                    if (glb_get(CALLWAIT+subs) <> 0)
                         # reset call waiting request
                         glb_set(CALLWAIT+subs, "") ;

                         # delete our netline from
caller's conference
                         del_conf(call, netline, vpline,
LOGOFF) ;

                         voslog("sc_play('call waiting
tones'(1305).vox)") ;
                         sc_play(vpline, "PR1305.vox") ; #
three tones

                         # add our netline to caller's
conference
                         add_conf(call, netline, LOGOFF) ;
                    endif

                    # check for selected caller active
                    if
(glb_get(CALL1ACT+SUBSMAX*call+subs) <> 0)
                         # selected caller active
                         # check for unselected caller
inactive
                         if
(glb_get(CALL1ACT+SUBSMAX*modulo2(call)+subs) eq 0)
                              # selected caller
active/unselected caller inactive
                              # set unselected forwarding
subscriber inactive
                              glb_set(FWRD1SUB+SUBSMAX
```

```
*modulo2(call)+subs, "") ;

                        # release unselected caller's
conference if unused
                        rel_conf(modulo2(call)) ;
                    endif
                else
                    # selected caller inactive
                    # check for unselected caller
inactive
                    if
(glb_get(CALL1ACT+SUBSMAX*modulo2(call)+subs) eq 0)
                        # both callers inactive
                        # set both forwarding
subscribers inactive
                        glb_set(FWRD1SUB+subs, "") ;
                        glb_set(FWRD2SUB+subs, "") ;

                        # release unselected caller's
conference if unused
                        rel_conf(modulo2(call)) ;
                        # release selected caller's
conference if unused
                        rel_conf(call) ;
                    else
                        # selected caller
inactive/unselected caller active
                        # check for three way
conference
                        if ((glb_get(FWRD1SUB+subs)
eq 0)
                         or (glb_get(FWRD2SUB+subs)
eq 0))
                            # three way conference
not active
                            # do NOT set selected
forwarding subscriber inactive here

                            voslog("TEST: Auto
switch conference request") ;

                            # switch to alternate
call
                            flash = "**" ;
                            goto switch_conf ;
                        else
                            # three way conference
active

                            # set selected
forwarding call inactive
```

```
                                glb_set(FWRD1ACT
+SUBSMAX*call+subs, "") ;

                                # set selected
forwarding subscriber inactive
                                glb_set(FWRD1SUB
+SUBSMAX*call+subs, "") ;

                                # release selected
caller's conference if unused
                                rel_conf(call) ;

                                # select active call
                                call = modulo2(call) ;
                            endif
                        endif
                    endif
                endif

                # check for forwarding tone input
                sc_clrdigits(vpline) ;
                code = sc_getdigits(vpline, 2, 3, 2) ;
                if (code eq T_MAXDT)
                    flash = sc_digits(vpline) ;

                    # check for priority call waiting
request
                    if (flash streq "##")
                        # set priority call waiting
request
                        glb_set(CALLWAIT+subs, -netline)
;
                        voslog("Priority call waiting
request") ;
                    endif

                    # check for forced voice mail request
                    if (flash streq "#9")
                        # check for caller active
                        if
(glb_get(CALL1ACT+SUBSMAX*call+subs) <> 0)
                            # start voicemail with
announcement
                            start_voicemail(call, 1) ;
                        endif
                        voslog("Forced voice mail
request") ;
                    endif
switch_conf:
                    # check for subscriber active on our
netline
```

```
                if ((glb_get(FWRD1SUB+subs) eq
netline)
                 or (glb_get(FWRD2SUB+subs) eq
netline))
                    # check for conference switch
request
                    if (flash streq "***")
                         # check for unselected call
active
                         # and three way conference
not active
                         if
((glb_get(CALL1ACT+SUBSMAX*modulo2(call)+subs) <> 0)
                         and ((glb_get(FWRD1SUB+subs)
eq 0)
                          or  (glb_get(FWRD2SUB+subs)
eq 0)))
                             voslog("Switch
conference request") ;
                             # reset call waiting
request
                             glb_set(CALLWAIT+subs,
"") ;
                             # delete our netline
from selected caller's conference
                             del_conf(call, netline,
vpline, LOGON) ;
                             # select next caller
                             call = modulo2(call) ;
                             # set selected
forwarding subscriber active
                             glb_set(FWRD1SUB
+SUBSMAX*call+subs, netline) ;
                             # set unselected
forwarding subscriber inactive
                             glb_set(FWRD1SUB
+SUBSMAX*modulo2(call)+subs, "") ;
                             # release unselected
caller's conference if unused
                             rel_conf(modulo2(call))
;
                             # stop selected call's
forwarding
                             stop_forwarding(call, 0)
;
```

```
                                # (FUTURE PROGRAMMABLE
OPTION)
                                # stop selected call's
voice mail
                                stop_voicemail(call) ;
                                # request answer
supervision be returned to selected caller
                                glb_set(CALL1ANS
+SUBSMAX*call+subs, vpline) ;
                                # set selected call's
forwarding answered
                                glb_set(FWRD1ANS
+SUBSMAX*call+subs, vpline) ;
                                # abort selected
caller's possible sc_play in progress
                                sc_abort(glb_get
(CALL1ACT+SUBSMAX*call+subs)) ;
                                voslog("sc_abort
("&glb_get(CALL1ACT+SUBSMAX*call+subs)&")") ;
                                # add our vpline to
selected conference
                                add_conf(call, vpline,
LOGOFF) ;
                                # play joining
conference tones
                                sc_playtone(vpline,
1000, 2000, -30, -30, 100) ;
                                voslog("sc_playtone(join
conf)") ;
                                # delete our vpline from
selected conference
                                del_conf(call, vpline,
netline, LOGOFF) ;
                                # add our netline to
selected caller's conference
                                add_conf(call, netline,
LOGON) ;
                            endif
                        endif
                        # check for conference switch
request
                        if (flash streq "#3")
```

```
active
not active
<> 0)
<> 0)
eq 0)
eq 0)))
conference request") ;

request
"") ;

supervision be returned to both
+SUBSMAX*call+subs, vpline) ;
+SUBSMAX*modulo2(call)+subs, vpline) ;

forwarding
;

OPTION)
mail
(call)) ;

subscribers active
netline) ;
netline) ;

calls answered
vpline) ;
```

```
# check for both callers
# and three way conference
if   ((glb_get(CALL1ACT+subs)
and   (glb_get(CALL2ACT+subs)
and  ((glb_get(FWRD1SUB+subs)
 or   (glb_get(FWRD2SUB+subs)
    voslog("Three way

    # reset call waiting
    glb_set(CALLWAIT+subs,

    # request answer
callers
    glb_set(CALL1ANS
    glb_set(CALL1ANS
    # stop selected call's
    stop_forwarding(call, 0)

    # (FUTURE PROGRAMMABLE
    # stop unselected voice
    stop_voicemail(modulo2

    # set both forwarding
    glb_set(FWRD1SUB+subs,
    glb_set(FWRD2SUB+subs,

    # set both forwarding
    glb_set(FWRD1ANS+subs,
    glb_set(FWRD2ANS+subs,
```

```
vpline) ;

                                # set both forwarding
netlines
                                glb_set(FWRD1NET+subs,
netline) ;
                                glb_set(FWRD2NET+subs,
netline) ;

                                # delete unselected
caller's netline from his conference
                                del_conf(modulo2(call),
glb_get(CALL1NET+SUBSMAX*modulo2(call)+subs),
glb_get(CALL1ACT+SUBSMAX*modulo2(call)+subs), LOGON) ;

                                # get unselected
caller's conference number
                                conf =
glb_get(CALL1CNF+SUBSMAX*modulo2(call)+subs) ;

                                # check for using active
specified conference
                                if ((conf > 0) and (conf
<= CONFMAX))
                                    # check if
unselected caller's conference still in use
                                    if
((glb_get(BARG1SUB+SUBSMAX*modulo2(call)+subs) eq 0)
                                    and
(glb_get(MAIL1ACT+SUBSMAX*modulo2(call)+subs) eq 0))
                                        # request
conference deallocation
                                        msg_put
(mainpid, "E"&conf) ;
                                        if ((code =
msg_get(3)) strneq "E-OK")
                                            voslog
("ERROR: code "&code&" when deallocating conference
"&conf) ;
                                        else
                                            voslog
("Deallocating conference "&conf) ;
                                            # set
unselected caller's conference inactive
                                            glb_set
(CALL1CNF+SUBSMAX*modulo2(call)+subs, "") ;
                                        endif
                                    endif
                                else
                                    voslog("ERROR:
deallocating inactive conference") ;
```

```
                                endif

                                # abort unselected
caller's possible sc_play in progress
                                sc_abort(glb_get
(CALL1ACT+SUBSMAX*modulo2(call)+subs)) ;
                                voslog("sc_abort
("&glb_get(CALL1ACT+SUBSMAX*modulo2(call)+subs)&")") ;

                                # add our vpline to
selected conference
                                add_conf(call, vpline,
LOGOFF) ;

                                # play joining
conference tones
                                sc_playtone(vpline,
1000, 2000, -30, -30, 100) ;
                                voslog("sc_playtone(join
conf)") ;

                                # delete our vpline from
selected conference
                                del_conf(call, vpline,
netline, LOGOFF) ;

                                # add unselected
caller's netline to selected caller's conference
                                add_conf(call,
glb_get(CALL1NET+SUBSMAX*modulo2(call)+subs), LOGON) ;

                                # check for unselected
conference inactive
                                if
(glb_get(CALL1CNF+SUBSMAX*modulo2(call)+subs) strneq "")
                                    voslog("TEST:
Deselected conference
"&glb_get(CALL1CNF+SUBSMAX*modulo2(call)+subs)&" not
inactive") ;
                                endif

                                # update unselected
caller's conference to selected caller's conference
                                glb_set(CALL1CNF
+SUBSMAX*modulo2(call)+subs,
glb_get(CALL1CNF+SUBSMAX*call+subs)) ;
                            endif
                        endif
                    endif
                endif
```

```
                # check for subscriber active on our
netline
                if ((glb_get(FWRD1SUB+subs) eq netline)
                 or (glb_get(FWRD2SUB+subs) eq netline))
                    # check for either caller active
                    if
((glb_get(CALL1ACT+SUBSMAX*call+subs) eq 0)
                    and
(glb_get(CALL1ACT+SUBSMAX*modulo2(call)+subs) eq 0))
                        hangup() ;
                        break ;
                    endif
                else
                    # check for caller and barger not
active on this call
                    if
((glb_get(CALL1ACT+SUBSMAX*call+subs) eq 0)
                    and
(glb_get(BARG1SUB+SUBSMAX*call+subs) eq 0))
                        hangup() ;
                        break ;
                    endif
                endif
            endfor
        endfor
end

onsignal
    voslog("Onsignal") ;
    hangup() ;
    restart ;
end

func hangup()
    # abort our multitasking calls in progress
    sc_abort(vpline) ;

    # wait for all our multitasking calls to finish
    for (code = 1; (sc_stat(vpline) <> 0) and (code <=
20); code++)
        sleep(5) ;
    endfor
    if (sc_stat(vpline) <> 0)
        voslog("ERROR: 10 seconds on vpline "&vpline&"
without idle status") ;
    endif

    # put vpline onhook
    while ((code = sc_onhook(vpline)) <> T_ONH)
        voslog("ERROR: code "&code&" while putting
vpline "&vpline&" onhook") ;
```

```
        sleep(10) ;
    endwhile
    voslog("sc_onhook("&vpline&")") ;

    # delete vpline from our conference
    del_conf(call, vpline, netline, LOGOFF) ;

    # delete netline from our conference
    del_conf(call, netline, vpline, LOGON) ;

    # remove vpline receive from netline transmit
    if ((code = Sbrk(vpline, netline, SSAS_ONHOOK,
SSAL_NO)) <> 0)
        voslog("Sbrk("&vpline&","&netline&") code
"&code&" error") ;
    else
        voslog("Sbrk("&vpline&","&netline&"):
"&Sstatus(vpline)) ;
    endif

    if ((netline >= NETBGN) and (netline <= NETEND))
        # remove netline receive from vpline transmit
        # and enable netline SS96 alerting
        if ((code = Sbrk(netline, vpline, SSAS_ONHOOK,
SSAL_YES)) <> 0)
            voslog("Sbrk("&netline&","&vpline&") code
"&code&" error") ;
        else
            voslog("Sbrk("&netline&","&vpline&"):
"&Sstatus(netline)) ;
        endif
    else
        # remove netline receive from vpline transmit
        # and disable netline SS96 alerting
        if ((code = Sbrk(netline, vpline, SSAS_ONHOOK,
SSAL_NO)) <> 0)
            voslog("Sbrk("&netline&","&vpline&") code
"&code&" error") ;
        else
            voslog("Sbrk("&netline&","&vpline&"):
"&Sstatus(netline)) ;
        endif
    endif

    # release netline/vpline allocations
    dealloc() ;

    # check for active call
    if (call strneq "")
        # check for forwarding subscriber answered on
our netline
```

```
          if ((glb_get(FWRD1SUB+subs) eq netline)
           or (glb_get(FWRD2SUB+subs) eq netline))
               # set forwarding subscribers inactive
               glb_set(FWRD1SUB+subs, "") ;
               glb_set(FWRD2SUB+subs, "") ;
          endif

          # set our netline inactive
          glb_set(FWRD1NET+SUBSMAX*call+subs, "") ;

          # set our call progress inactive
          glb_set(FWRD1CPA+SUBSMAX*call+subs, "") ;

          # check for forwarding answered
          if (glb_get(FWRD1ANS+SUBSMAX*call+subs) <> 0)
               # set forwarding task inactive and stopped
(do last)
               glb_set(FWRD1ACT+SUBSMAX*call+subs, 0) ;
          else
               # set forwarding task inactive (do last)
               glb_set(FWRD1ACT+SUBSMAX*call+subs, "") ;
          endif
     endif

     # release caller's conference if unused
     rel_conf(call) ;

     # set task debug screen status
     scr_stat("Idle") ;
endfunc

func dealloc()
     # release netline/vpline allocations
     msg_put(mainpid, "D"&netline&vpline) ;
     if ((code = msg_get(3)) strneq "D-OK")
          voslog("ERROR: code "&code&" when deallocating
netline "&netline) ;
     endif
endfunc

func stop_forwarding(scall, sforce)

     # check for not forced stop request
     # and for forwarding answered
     # and for not "transparent" forwarding mode
     if ((sforce eq 0)
     and (glb_get(FWRD1ANS+SUBSMAX*scall+subs) <> 0)
     and (glb_get(FORWARD+subs) strneq
glb_get(CALLER_DID+subs)))
          return;
     endif
```

```
    sem_set(SEM_FWRD+subs) ;

    # check for forwarding active
    if (glb_get(FWRD1ACT+SUBSMAX*scall+subs) <> 0)
         voslog("stop_forwarding()") ;

         # delete forwarding netline from caller's
conference
         del_conf(scall,
glb_get(FWRD1NET+SUBSMAX*scall+subs),
glb_get(FWRD1ACT+SUBSMAX*scall+subs), LOGON) ;

         # set call answered so forward task marks
itself inactive and stopped
         glb_set(FWRD1ANS+SUBSMAX*call+subs, vpline) ;

         # abort forwarding task
         sc_hangup(glb_get(FWRD1ACT+SUBSMAX*scall+subs))
;
         voslog("sc_hangup("&glb_get(FWRD1ACT+SUBSM
AX*scall+subs)&")") ;

         # set forwarding task inactive and stopped
         glb_set(FWRD1ACT+SUBSMAX*scall+subs, 0) ;
     endif
     sem_clear(SEM_FWRD+subs) ;
endfunc

func start_voicemail(scall, smsg)

     # check for voice mail active
     if (glb_get(MAIL1ACT+SUBSMAX*scall+subs) eq 0)
          # request voice mail vpline task be started
          msg_put(mainpid, "V"&scall&subs) ;
          code = msg_get(3) ;
          if (substr(code, 1, 1) strneq "V")
               voslog("ERROR: code "&code&" when
allocating voice mail vpline") ;
          else
               # set voice mail task active
               glb_set(MAIL1ACT+SUBSMAX*scall+subs,
substr(code, 2, 99)) ;

               if (smsg <> 0)
                    # add our vpline to conference
                    add_conf(scall, vpline, LOGOFF) ;

                    # play 'connecting to voice mail'
message
                    voslog("sc_play('to voice
mail'(030).vox)") ;
```

```
                    sc_play(vpline, "PR0030.vox") ;

                    # delete our vpline from conference
                    del_conf(scall, vpline, netline,
LOGOFF) ;
               endif
          endif
     endif
endfunc

func stop_voicemail(scall)

     sem_set(SEM_MAIL+subs) ;

     # check for voice mail active
     if (glb_get(MAIL1ACT+SUBSMAX*scall+subs) <> 0)
          voslog("stop_voicemail()") ;

          # delete voice mail netline from caller's
conference
          del_conf(scall,
glb_get(MAIL1NET+SUBSMAX*scall+subs),
glb_get(MAIL1ACT+SUBSMAX*scall+subs), LOGON) ;

          # abort voice mail task
          sc_hangup(glb_get(MAIL1ACT+SUBSMAX*scall+subs))
;
          voslog("sc_hangup("&glb_get(MAIL1ACT+SUBSM
AX*scall+subs)&")") ;

          # set voice mail task inactive and stopped
          glb_set(MAIL1ACT+SUBSMAX*scall+subs, 0) ;
     endif
     sem_clear(SEM_MAIL+subs) ;
endfunc

func modulo2(scall)
     return   ((scall+1)-(scall+1)/2*2) ;
endfunc

     include "confsubs.inc"     # conferencing
subroutines
```

APPENDIX IV

SOURCE CODE LISTING OF A PAGE TASK
USED BY MAIN PROGRAM

```
#    PAGE.VS - D121 line task
#
#    Designed to work in concert with MAIN.VS
#
#-------------------------------------------------------
------------------------

dec
     include "datmod.inc"        # SS96 library
definitions
     include "global.inc"        # define global variable
numbers

     var code:16 ;          # return value from function
calls
     var vpline:3 ;
     var netline:3 ;
     var mainpid:3 ;
     var subs:3 ;           # subscriber number
     var sdate:7 ;          # starting date
     var stime:7 ;          # starting time

     var line:3 ;           # needed for scr_stat
end

program

# Initializations
     vpline = arg() ;
     line = vpline ;                  # debug only
     if (glb_get(MAINFLG) <> 99)          # if first time
initialization
          mainpid = glb_get(MAINPID) ;

          # inform main task of our pid
          msg_put(mainpid, getpid()) ;

          # wait for main task to be ready to go
          while (glb_get(MAINFLG) <> 99)
          endwhile
     endif

# Begin call processing
     sc_use(vpline) ;
```

```
# Set minimum detected wink duration to 50 msec (from
100ms default)
     sc_setparm(vpline, 518, 5) ;

# Set maximum detected wink duration to 300 msec (from
200ms default)
     sc_setparm(vpline, 519, 30) ;

# Turn on speech card function calls trace
#    sc_trace(vpline, 1) ;

# Attempt to go on-hook
     sc_watch(vpline, "+-", 1) ;
     while ((code = sc_onhook(vpline)) <> T_ONH)
          voslog("ERROR: "&code&" - onhook failure") ;
          sleep(10) ;
     endwhile

# Wait for message from main task indicating start of
call
     for (;;)
          # ignore disconnects
          sc_watch(vpline, "lw+-", 1) ;

          while (length(vpline = msg_get(3600)) eq 0)
          endwhile
          if ((vpline < PAGEBGN) or (vpline > PAGEEND))
               voslog("ERROR: invalid vpline number
"&vpline&" message received") ;
               continue ;
          endif

          while (length(subs = msg_get(3)) eq 0)
          endwhile
          if ((subs < 0) or (subs >= SUBSMAX))
               voslog("ERROR: invalid subscriber number
"&subs&" message received") ;
               continue ;
          endif

          # ask alternate outbound netline be allocated
          msg_put(mainpid, "A"&vpline) ;
          if ((code = msg_get(3)) strneq "A-OK")
               voslog("ERROR: code "&code&" when
allocating outbound netline") ;
               continue;
          endif

          # get allocated netline
          while (length(netline = msg_get(3)) eq 0)
```

```
        endwhile
        if ((netline < ALTBGN) or (netline > ALTEND))
            voslog("ERROR: invalid netline number
"&netline&" message") ;
            # release netline/vpline allocations
            dealloc() ;
            continue ;
        endif

        #defer hang up jump to onsignal
        sc_sigctl("d") ;

        # drive vpline receive from netline transmit
        if ((code = Scon(vpline, netline, SSAS_PASS,
SSAL_NO)) <> 0)
            voslog("Scon("&vpline&","&netline&") code
"&code&" error") ;
        else
            voslog("Scon("&vpline&","&netline&"):
"&Sstatus(vpline)) ;
        endif

        # drive netline receive from vpline transmit
        if ((code = Scon(netline, vpline, SSAS_PASS,
SSAL_NO)) <> 0)
            voslog("Scon("&netline&","&vpline&") code
"&code&" error") ;
        else
            voslog("Scon("&netline&","&vpline&"):
"&Sstatus(netline)) ;
        endif

        #throw away deferred hang ups
        sc_sigctl("c") ;

        # set task debug screen status
        scr_stat("P-"&netline) ;

        # save the initial time
        sdate = date() ;
        stime = time() ;
dial_page:
        # go off hook
        if ((code = sc_offhook(vpline)) <> T_OFFH)
            voslog("ERROR: code "&code&" while putting
vpline "&vpline&" offhook") ;
            goto retry ;
        else
            voslog("sc_offhook("&vpline&")") ;
        endif
```

```
          # wait one second for next event report
          code = sc_wait(vpline, 10) ;
          voslog("sc_wait("&vpline&",10) = "&code) ;

          # check for loop current event report
          if (code <> T_LCON)
               voslog("ERROR: Loop current timeout on
vpline "&vpline) ;
               goto retry ;
          endif

          # wait one second for next event report
          code = sc_wait(vpline, 10) ;
          voslog("sc_wait("&vpline&",10) = "&code) ;

          # check for wink event report
          if (code <> T_WKRECV)
               voslog("ERROR: Wink timeout on vpline
"&vpline) ;
               goto retry ;
          endif

          voslog("Dialing pager "&glb_get(PAGER+subs)) ;

          # call the paging system - wait for completion
          if ((code = sc_dial(vpline,
glb_get(STRATUS_PRE)&glb_get(PAGER+subs)&glb_get
(STRATUS_PST))) <> T_DIAL)
               voslog("ERROR: code "&code&" while dialing
pager") ;
               goto retry ;
          endif

          # wait 18 seconds for next event report
          code = sc_wait(vpline, 180) ;
          voslog("sc_wait("&vpline&",180) = "&code) ;

          # check for loop current event report
          if (code <> T_LCON)
               voslog("ERROR: Answer supervision timeout
on vpline "&vpline) ;
retry:
               if (timesub(date(), time(), sdate, stime)
<= 10)
                    # put vpline onhook
                    if ((code = sc_onhook(vpline)) <>
T_ONH)
                         voslog("ERROR: code "&code&"
while putting vpline "&vpline&" onhook") ;
                    else
                         voslog("sc_onhook("&vpline&")") ;
```

```
                endif

                # wait for onhook recognition
                sleep(10) ;

                goto dial_page ;
            else
                # disconnect
                hangup() ;
                # look for next page request
                continue ;
            endif
        else
            voslog("Answer supervision") ;
        endif

        # now watch for disconnect
        sc_watch(vpline, "dl+-", 1) ;

        # delay for paging system to give proceed
signal
        sleep(20) ;

        voslog("Sending to pager caller number
"&glb_get(CALLER_NUMB+subs)) ;

        # enter caller's number into paging system
        while ((code = sc_dial(vpline,
glb_get(CALLER_NUMB+subs))) <> T_DIAL)
            voslog("ERROR: code "&code&" while sending
Caller ID to paging system") ;
            sleep(10) ;
        endwhile

        # delay for paging system to disconnect (TEST
ONLY?)
        sleep(40) ;

        # disconnect
        hangup() ;
    endfor
end

onsignal
    voslog("Onsignal") ;
    hangup() ;
    restart ;
end

func hangup()
```

```
    # abort our multitasking calls in progress
    sc_abort(vpline) ;

    # wait for all our multitasking calls to finish
    for (code = 1; (sc_stat(vpline) <> 0) and (code <=
20); code++)
        sleep(5) ;
    endfor
    if (sc_stat(vpline) <> 0)
        voslog("ERROR: 10 seconds on vpline "&vpline&"
without idle status") ;
    endif

    # put vpline onhook
    while ((code = sc_onhook(vpline)) <> T_ONH)
        voslog("ERROR: code "&code&" while putting
vpline "&vpline&" onhook") ;
        sleep(10) ;
    endwhile
    voslog("sc_onhook("&vpline&")") ;

    # remove vpline receive from netline transmit
    if ((code = Sbrk(vpline, netline, SSAS_ONHOOK,
SSAL_NO)) <> 0)
        voslog("Sbrk("&vpline&","&netline&") code
"&code&" error") ;
    else
        voslog("Sbrk("&vpline&","&netline&"):
"&Sstatus(vpline)) ;
    endif

    if ((netline >= NETBGN) and (netline <= NETEND))
        # remove netline receive from vpline transmit
        # and enable netline SS96 alerting
        if ((code = Sbrk(netline, vpline, SSAS_ONHOOK,
SSAL_YES)) <> 0)
            voslog("Sbrk("&netline&","&vpline&") code
"&code&" error") ;
        else
            voslog("Sbrk("&netline&","&vpline&"):
"&Sstatus(netline)) ;
        endif
    else
        # remove netline receive from vpline transmit
        # and disable netline SS96 alerting
        if ((code = Sbrk(netline, vpline, SSAS_ONHOOK,
SSAL_NO)) <> 0)
            voslog("Sbrk("&netline&","&vpline&") code
"&code&" error") ;
        else
            voslog("Sbrk("&netline&","&vpline&"):
```

```
"&Sstatus(netline)) ;
        endif
    endif
    # release netline/vpline allocations
    dealloc() ;

    # set task debug screen status
    scr_stat("Idle") ;
endfunc

func dealloc()
    # release netline/vpline allocations
    msg_put(mainpid, "D"&netline&vpline) ;

    if ((code = msg_get(3)) strneq "D-OK")
        voslog("ERROR: code "&code&" when deallocating
netline "&netline) ;
    endif
endfunc
```

APPENDIX V

SOURCE CODE LISTING OF A VOICE MAIL TASK
USED BY THE MAIN PROGRAM

```
#
#    VMAIL.VS - D121 line task
#
#    Designed to work in concert with MAIN.VS
#
#-------------------------------------------------------
-----------------------

dec
     include "datmod.inc"        # SS96 library
definitions
     include "global.inc"        # define global variable
numbers

     var code:16 ;           # return value from function
calls
     var vpline:3 ;
     var netline:3 ;
     var mainpid:3 ;
     var subs:3 ;            # subscriber number
     var call:3 ;            # call number
     var conf:3 ;            # conference number
     var sdate:7 ;           # starting date
     var stime:7 ;           # starting time

     var stats:16 ;              # beginning netline
status (confsubs)
     var line:3 ;            # needed for scr_stat
end

program

# Initializations

     vpline = arg() ;
     line = vpline ;                    # debug only
     if (glb_get(MAINFLG) <> 99)          # if first time
initialization
          mainpid = glb_get(MAINPID) ;
          msg_put(mainpid, getpid()) ;    # inform main
task of our pid

          # wait for main task to be ready to go
          while (glb_get(MAINFLG) <> 99)
```

```
        endwhile
    endif

# Begin call processing
    sc_use(vpline) ;

# Set minimum detected wink duration to 50 msec (from
100ms default)
    sc_setparm(vpline, 518, 5) ;

# Set maximum detected wink duration to 300 msec (from
200ms default)
    sc_setparm(vpline, 519, 30) ;

# Turn on speech card function calls trace
#   sc_trace(vpline, 1) ;

# Wait for onhook complete
    sc_watch(vpline, "+-", 1) ;
    while ((code = sc_onhook(vpline)) <> T_ONH)
        voslog("ERROR: code "&code&" - onhook failure")
;
        sleep(10) ;
    endwhile

# Wait for message from main task indicating start of
call
    for (;;)
        # initialize netline in case of reorder
        netline = 0 ;

        # ignore disconnect
        sc_watch(vpline, "lw+-", 1) ;

        while (length(vpline = msg_get(3600)) eq 0)
        endwhile
        if ((vpline < MAILBGN) or (vpline > MAILEND))
            voslog("ERROR: invalid vpline number
"&vpline&" message received") ;
            # set voice mail inactive and stopped
# (call not set
yet) glb_set(MAIL1ACT+SUBSMAX*call+subs, 0) ;
            continue ;
        endif

        # get subscriber number
        while (length(subs = msg_get(3)) eq 0)
        endwhile
        if ((subs < 0) or (subs >= SUBSMAX))
            voslog("ERROR: invalid subscriber number
"&subs&" message received") ;
```

```
            # set voice mail inactive and stopped
# (call not set
yet) glb_set(MAIL1ACT+SUBSMAX*call+subs, 0) ;
            continue ;
        endif

        # get call number
        while (length(call = msg_get(3)) eq 0)
        endwhile
        if ((call < 0) or (call >= CALLMAX))
            voslog("ERROR: invalid call number
"&call&" message received") ;
            # set voice mail inactive and stopped
# (call #
invalid) glb_set(MAIL1ACT+SUBSMAX*call+subs, 0) ;
            continue ;
        endif

        # request outbound netline allocation
        msg_put(mainpid, "A"&vpline) ;
        if ((code = msg_get(3)) strneq "A-OK")
            voslog("ERROR: code "&code&" when
allocating outbound netline") ;

            play_reorder(call) ;

            # set voice mail inactive and stopped
            glb_set(MAIL1ACT+SUBSMAX*call+subs, 0) ;

            # release caller's conference if unused
            rel_conf(call) ;

            # get next vmail task request
            continue ;
        endif

        # get allocated netline
        while (length(netline = msg_get(3)) eq 0)
        endwhile
        if ((netline < ALTBGN) or (netline > ALTEND))
            voslog("ERROR: invalid netline number
"&netline&" message") ;

            play_reorder(call) ;

            # release netline/vpline allocations
            dealloc() ;

            # set voice mail inactive and stopped
            glb_set(MAIL1ACT+SUBSMAX*call+subs, 0) ;
```

```
            # release caller's conference if unused
            rel_conf(call) ;

            # get next vmail task request
            continue ;
        endif

        # set voice mail netline
        glb_set(MAIL1NET+SUBSMAX*call+subs, netline) ;

        # defer hang up jump to onsignal
        sc_sigctl("d") ;

        # drive alternate netline receive from vpline
transmit
        if ((code = Scon(netline, vpline, SSAS_PASS,
SSAL_NO)) <> 0)
            voslog("Scon("&netline&","&vpline&") code
"&code&" error") ;
        else
            voslog("Scon("&netline&","&vpline&"):
"&Sstatus(netline)) ;
        endif

        # drive vpline receive from alternate netline
transmit
        if ((code = Scon(vpline, netline, SSAS_PASS,
SSAL_NO)) <> 0)
            voslog("Scon("&vpline&","&netline&") code
"&code&" error") ;
        else
            voslog("Scon("&vpline&","&netline&"):
"&Sstatus(vpline)) ;
        endif

        # throw away deferred hang ups
        sc_sigctl("c") ;

        # set task debug screen status
        scr_stat("V-"&netline) ;

        # save the initial time
        sdate = date() ;
        stime = time() ;
dial_mail:
        # go off hook
        if ((code = sc_offhook(vpline)) <> T_OFFH)
            voslog("ERROR: code "&code&" while putting
vpline "&vpline&" offhook") ;
            goto retry ;
        else
```

```
            voslog("sc_offhook("&vpline&")") ;
        endif

        # wait one second for next event report
        code = sc_wait(vpline, 10) ;
        voslog("sc_wait("&vpline&",10) = "&code) ;

        # check for loop current event report
        if (code <> T_LCON)
            voslog("ERROR: Loop current timeout on
vpline "&vpline) ;
            goto retry ;
        endif

        # wait one second for next event report
        code = sc_wait(vpline, 10) ;
        voslog("sc_wait("&vpline&",10) = "&code) ;

        # check for wink event report
        if (code <> T_WKRECV)
            voslog("ERROR: Wink timeout on vpline
"&vpline) ;
            goto retry ;
        endif

        voslog("Dialing voice mail number
"&glb_get(VOICEMAIL+subs)) ;

        # dial the voice mail number - wait until
dialing complete
        if ((code = sc_dial(vpline,
glb_get(STRATUS_PRE)&glb_get(VOICEMAIL+subs)&glb_get
(STRATUS_PST))) <> T_DIAL)
            voslog("ERROR: code "&code&" while dialing
voice mail number") ;
            goto retry ;
        endif

        # add voice mail netline to current caller's
conference
        add_conf(call, netline, 1) ;

        # wait 18 seconds for next event report
        code = sc_wait(vpline, 180) ;
        voslog("sc_wait("&vpline&",180) = "&code) ;

        # check for loop current event report
        if (code <> T_LCON)
            voslog("ERROR: Answer supervision timeout
on vpline "&vpline) ;
```

```
            # delete netline from our conference
            del_conf(call, netline, vpline, 1) ;
retry:
            if (timesub(date(), time(), sdate, stime)
<= 10)
                # put vpline onhook
                if ((code = sc_onhook(vpline)) <>
T_ONH)
                    voslog("ERROR: code "&code&"
while putting vpline "&vpline&" onhook") ;
                else
                    voslog("sc_onhook("&vpline&")") ;
                endif

                # wait for onhook recognition
                sleep(10) ;

                goto dial_mail ;
            else
                # disconnect and look for next vmail
message
                hangup() ;
                continue ;
            endif
        else
            voslog("Answer supervision") ;
        endif

        # request answer supervision be returned to
caller
        glb_set(CALL1ANS+SUBSMAX*call+subs, vpline) ;

        # now watch for disconnect
        sc_watch(vpline, "dl+-", 1) ;

        for (;;)
            # check for caller inactive
            # and barger inactive
            # and forwarding inactive
            if (((call >= CALLMAX) or
(glb_get(CALL1ACT+SUBSMAX*call+subs) eq 0))
            and ((call >= BARGMAX) or
(glb_get(BARG1SUB+SUBSMAX*call+subs) eq 0))
            and ((call >= FWRDMAX) or
(glb_get(FWRD1ACT+SUBSMAX*call+subs) eq 0)))
                hangup() ;
                break ;
            endif
        endfor
    endfor
end
```

```
onsignal
     voslog("Onsignal") ;
     hangup() ;
     restart ;
end

func hangup()
     # abort our multitasking calls in progress
     sc_abort(vpline) ;

     # wait for all our multitasking calls to finish
     for (code = 1; (sc_stat(vpline) <> 0) and (code <=
20); code++)
          sleep(5) ;
     endfor
     if (sc_stat(vpline) <> 0)
          voslog("ERROR: 10 seconds on vpline "&vpline&"
without idle status") ;
     endif

     # put vpline onhook
     while ((code = sc_onhook(vpline)) <> T_ONH)
          voslog("ERROR: code "&code&" while putting
vpline "&vpline&" onhook") ;
          sleep(10) ;
     endwhile
     voslog("sc_onhook("&vpline&")") ;

     # delete netline from our conference
     del_conf(call, netline, vpline, 1) ;

     # remove vpline receive from netline transmit
     if ((code = Sbrk(vpline, netline, SSAS_ONHOOK,
SSAL_NO)) <> 0)
          voslog("Sbrk("&vpline&","&netline&") code
"&code&" error") ;
     else
          voslog("Sbrk("&vpline&","&netline&"):
"&Sstatus(vpline)) ;
     endif

     if ((netline >= NETBGN) and (netline <= NETEND))
          # remove netline receive from vpline transmit
          # and enable netline SS96 alerting
          if ((code = Sbrk(netline, vpline, SSAS_ONHOOK,
SSAL_YES)) <> 0)
               voslog("Sbrk("&netline&","&vpline&") code
"&code&" error") ;
          else
               voslog("Sbrk("&netline&","&vpline&"):
```

```
"&Sstatus(netline)) ;
          endif
     else
          # remove netline receive from vpline transmit
          # and disable netline SS96 alerting
          if ((code = Sbrk(netline, vpline, SSAS_ONHOOK,
SSAL_NO)) <> 0)
               voslog("Sbrk("&netline&","&vpline&") code
"&code&" error") ;
          else
               voslog("Sbrk("&netline&","&vpline&"):
"&Sstatus(netline)) ;
          endif
     endif

     # release netline/vpline allocations
     dealloc() ;

     if (call strneq "")
          # set voice mail netline inactive
          glb_set(MAIL1NET+SUBSMAX*call+subs, "") ;

          # check for request for voicemail to not be
marked as not having run
          if (glb_get(MAIL1ACT+SUBSMAX*call+subs) strneq
"")
               # set voice mail inactive but stopped (do
last)
               glb_set(MAIL1ACT+SUBSMAX*call+subs, 0) ;
          endif
     endif

     # release selected conference if unused
     rel_conf(call) ;

     # set task debug screen status
     scr_stat("Idle") ;
endfunc

func dealloc()
     # release netline/vpline allocations
     msg_put(mainpid, "D"&netline&vpline) ;
     if ((code = msg_get(3)) strneq "D-OK")
          voslog("ERROR: code "&code&" when deallocating
netline "&netline) ;
     endif
endfunc

func play_reorder(scall)

     # add our vpline to caller's conference
```

```
    add_conf(scall, vpline, 0) ;

    voslog("sc_playtone(reorder)") ;
    # play reorder tones for 20 seconds
    for (code = 1; code <= 40; code++)
        sc_playtone(vpline, 480, 620, -24, -24, 30) ;
        # delay between tones
        sleep(2) ;
    endfor

    # delete our vpline from caller's conference
    del_conf(scall, vpline, netline, 0) ;
endfunc

func modulo2(scall)

return   ((scall+1)-(scall+1)/2*2) ;

endfunc

    include "confsubs.inc"      # conferencing
subroutines
```

APPENDIX VI

SOURCE CODE LISTING USED BY INBOUND TASK
FOR PHONE NUMBER PLAYBACK

```
#    spk_9: Speak one digit
#
#    ARGUMENT
#    a_num     Digit to speak 0..9,*,#
#
#    RETURNS
#    Return code from sc_play
#
#    No attempt is made to validate the a_num argument.

func spk_9(a_num)
     if (a_num streq "*")
          return sc_play(line, "c:\vos\src\STAR.vox");
     endif
     if (a_num streq "#")
          return sc_play(line, "c:\vos\src\POUND.vox");
     endif
     return sc_play(line,
"c:\vos\src\PR0"&(189+a_num)&".vox");
end
```

APPENDIX VII

SUBROUTINE LISTING DEFINING CONSTANTS
AND VARIABLES USED BY ALL TASKS

```
#    GLOBAL.INC

# Global variables

     const SUBSMAX = 7 ;          # number of subscribers
     const CALLMAX = 4 ;          # maximum inbound public
calls per subscriber
     const BARGMAX = 2 ;          # maximum active barger
calls per subscriber
     const FWRDMAX = 2 ;          # maximum forwarding
calls per subscriber
     const MAILMAX = 4 ;          # maximum voice mail
calls per subscriber
     const CONFMAX = 16 ;         # number of conferences
(16 x 6way = maximum)

     const NETBGN = 25 ;          # beginning PEB2 netline
channel
#    const NETEND = 28 ;          # ending PEB2 netline
channel (654-521x)
     const NETEND = 34 ;          # ending PEB2 netline
channel (654-578x)

     const ALTBGN = 73 ;          # beginning PEB4
alternate channel
     const ALTEND = 82 ;          # ending PEB4 alternate
channel

     const GDIDBGN = 1 ;          # beginning GETDID
vpline
     const GDIDEND = 4 ;          # ending GETDID vpline
     const FWRDBGN = 5 ;          # beginning FORWARD
vpline
     const FWRDEND = 8 ;          # ending FORWARD vpline
     const MAILBGN = 9 ;          # beginning VMAIL vpline
     const MAILEND = 10 ;         # ending VMAIL vpline
     const PAGEBGN = 11 ;         # beginning PAGE vpline
     const PAGEEND = 12 ;         # ending PAGE vpline

     const LOGOFF = 0 ;           # conference subroutines
voslog off
     const LOGON = 1 ;       # conference subroutines
voslog on
```

```
# Semaphore reference numbers

    const SEM_FWRD = 48 ;       # use when accessing
FWRDnACT (one per subscriber)
    const SEM_MAIL = SEM_FWRD+SUBSMAX ; # use when
accessing MAILnACT (one per subscriber)
    const SEM_next = SEM_MAIL+SUBSMAX ; # leave room
for SEM_MAIL+subs accesses
                        # maximum semaphore is 63

# Global memory reference numbers

    const MAINPID = 0 ;         # main task pid
    const MAINFLG = MAINPID+1 ;     # main task go-ahead
flag
    const STRATUS_PRE = MAINFLG+1 ; # Stratus dialing
preamble
    const STRATUS_PST = STRATUS_PRE+1 ;   # Stratus
dialing postamble
    const next = STRATUS_PST+1 ;    # room for more
globals (up to 1*SUBSMAX)

    const PSTN_PRE =     1*SUBSMAX ; # public network
dialing preamble
    const PSTN_PST =     2*SUBSMAX ; # public network
dialing postamble
    const FORWARD =      3*SUBSMAX ; # forwarding phone
number #1
    const PAGER =        4*SUBSMAX ; # pager phone
number
    const VOICEMAIL =    5*SUBSMAX ; # voice mail phone
number
    const CALLER_NUMB =  6*SUBSMAX ; # caller phone
number
    const CALLER_DID =   7*SUBSMAX ; # caller DID
    const BARGE_DID =    8*SUBSMAX ; # barger DID
    const PRGRM_PIN =    9*SUBSMAX ; # programing PIN
    const BARGE_PIN =   10*SUBSMAX ; # barge in PIN
    const CALLWAIT =    11*SUBSMAX ; # call waiting flag
    const CALL1ACT =    12*SUBSMAX ; # caller #1 active
(vpline)
    const CALL2ACT =    13*SUBSMAX ; # caller #2 active
(vpline)
    const CALL3ACT =    14*SUBSMAX ; # caller #3 active
(vpline)
    const CALL4ACT =    15*SUBSMAX ; # caller #4 active
(vpline)
    const CALL1CNF =    16*SUBSMAX ; # caller #1
conference
    const CALL2CNF =    17*SUBSMAX ; # caller #2
```

```
conference
     const CALL3CNF =    18*SUBSMAX ; # caller #3
conference
     const CALL4CNF =    19*SUBSMAX ; # caller #4
conference
     const CALL1ANS =    20*SUBSMAX ; # caller #1 answer
supervision (vpline)
     const CALL2ANS =    21*SUBSMAX ; # caller #2 answer
supervision (vpline)
     const CALL3ANS =    22*SUBSMAX ; # caller #3 answer
supervision (vpline)
     const CALL4ANS =    23*SUBSMAX ; # caller #4 answer
supervision (vpline)
     const CALL1NET =    24*SUBSMAX ; # caller #1 netline
     const CALL2NET =    25*SUBSMAX ; # caller #2 netline
     const CALL3NET =    26*SUBSMAX ; # caller #3 netline
     const CALL4NET =    27*SUBSMAX ; # caller #4 netline
     const BARG1SUB =    28*SUBSMAX ; # call #1 barger
subscriber active (netline)
     const BARG2SUB =    29*SUBSMAX ; # call #2 barger
subscriber active (netline)
     const BARG1ANS =    30*SUBSMAX ; # call #1 barger
answered (vpline)
     const BARG2ANS =    31*SUBSMAX ; # call #2 barger
answered (vpline)
     const FWRD1ACT =    32*SUBSMAX ; # call #1
forwarding active (vpline)
     const FWRD2ACT =    33*SUBSMAX ; # call #2
forwarding active (vpline)
     const FWRD1SUB =    34*SUBSMAX ; # call #1
forwarding subscriber active (netline)
     const FWRD2SUB =    35*SUBSMAX ; # call #2
forwarding subscriber active (netline)
     const FWRD1ANS =    36*SUBSMAX ; # call #1
forwarding answered (vpline)
     const FWRD2ANS =    37*SUBSMAX ; # call #2
forwarding answered (vpline)
     const FWRD1NET =    38*SUBSMAX ; # call #1
forwarding netline
     const FWRD2NET =    39*SUBSMAX ; # call #2
forwarding netline
     const FWRD1CPA =    40*SUBSMAX ; # call #1
forwarding call progress active
     const FWRD2CPA =    41*SUBSMAX ; # call #2
forwarding call progress active
     const MAIL1ACT =    42*SUBSMAX ; # call #1 voice
mail active (vpline)
     const MAIL2ACT =    43*SUBSMAX ; # call #2 voice
mail active (vpline)
     const MAIL3ACT =    44*SUBSMAX ; # call #3 voice
mail active (vpline)
```

```
    const MAIL4ACT =    45*SUBSMAX ; # call #4 voice
mail active (vpline)
    const MAIL1NET =    46*SUBSMAX ; # call #1 voice
mail netline
    const MAIL2NET =    47*SUBSMAX ; # call #2 voice
mail netline
    const MAIL3NET =    48*SUBSMAX ; # call #3 voice
mail netline
    const MAIL4NET =    49*SUBSMAX ; # call #4 voice
mail netline
    const GLOBALS  =    50*SUBSMAX ; # end of active
global

# Values for termination types (last_term) and event
type

    const  T_NOTERM  = 0 ;      # No termination
received
    const  T_MAXDT   = 1 ;      # Maximum DTMF digits
received
    const  T_TERMDT  = 2 ;      # Terminating DTMF digit
received
    const  T_STOP    = 3 ;      # User stop
    const  T_DOSERR  = 4 ;      # Dos error
    const  T_MAXBYT  = 5 ;      # Max bytes reached on
play or rec
    const  T_HFAIL   = 6 ;      # Hardware failure
    const  T_TIME    = 7 ;      # Multi-tasking function
timed out
    const  T_OFFH    = 8 ;      # Offhook complete
    const  T_DIAL    = 9 ;      # Dialing complete
    const  T_SIL     = 10 ;         # Maximum silence
received
    const  T_EOF     = 11 ;         # Eof reached on
playback
    const  T_LCTERM  = 12 ;         # Terminate by drop
in loop signal
    const  T_DFULL   = 13 ;         # Disk full
    const  T_ONH     = 14 ;         # Onhook complete
    const  T_MDTERM  = 17 ;         # AMX80 disconnect
termination
    const  T_CATERM  = 18 ;         # Call analysis
termination
    const  T_LCREV   = 19 ;         # Loop signal
battery reversal
    const  T_LC      = 20 ;         # Loop signal drop
event
    const  T_RING    = 21 ;         # Rings received
    const  T_SILOFF  = 22 ;         # Silence off
    const  T_SILON   = 23 ;         # Silence on
    const  T_AMXCON  = 24 ;         # AMX8x channel
```

```
connect
     const   T_AMXDIS   = 25 ;          # AMX8x channel
disconnect
     const   T_LCON     = 26 ;          # Loop signal on
event
     const   T_MAXRNG   = 27 ;          # Max rings reached
on AMX81 conn
     const   T_MCTERM   = 28 ;          # Rings termiated by
AMX8 connect
     const   T_MDTMF    = 29 ;          # Terminated by
masked DTMF digit
     const   T_IDTIME   = 30 ;          # Inter  digit delay
exceeded
     const   T_NSIL     = 31 ;          # Terminated by a
max non-silence
     const   T_BUFFUL   = 32 ;          # Termination from
EMS buffer full
     const   T_BUFEMP   = 33 ;          # Terminated from
EMS buffer empty
     const   T_EMSERR   = 34 ;          # Terminated by EMS
error
     const   T_EMSLOW   = 35 ;          # EMS buffer low
     const   T_EMSHI    = 36 ;          # EMS buffer high
     const   T_STPARM   = 37 ;          # Parameter updated
     const   T_WINK     = 38 ;          # Wink protocol
complete
     const   T_WKRECV   = 39 ;          # Wink received
     const   T_DTMF     = 40 ;          # DTMF digit
received
     const   T_TONEON   = 41 ;          # Tone on detect
     const   T_TONEOFF  = 42 ;          # Tone off detect
     const   T_BADTERM  = 43 ;          # Invalid
termination condition
     const   T_MTONEON  = 44 ;          # Terminated by Tone
On
     const   T_MTONEOFF= 45 ;           # Terminated by Tone
Off
     const   T_CAERROR  = 46 ;          # Call progress
error
     const   T_TGERR    = 47 ;          # Tone generation
template error
     const   T_TGCMPLT  = 48 ;          # Tone generation
complete
     const   T_ADSIERR  = 49 ;          # ADSI protocol
failure
     const   MAXTERM    = 49 ;          # Max(last)
termination type

# Call Analysis termination data types/call status
values
```

```
    const  CA_BUSY    = 7 ;      # Called line is busy
    const  CA_NOAN    = 8 ;      # Called line did not
answer
    const  CA_NORNG   = 9 ;      # Called line did not
ring
    const  CA_CONN    = 10 ;          # Called line
connected
    const  CA_OPINT   = 11 ;          # Called line recvd
oper intercept
    const  CA_NODIALTONE = 17 ;     # Called line recvd
no dialtone
    const  CA_FAXTONE = 18 ;    # Called line recvd fax
tone
```

APPENDIX VIII

SUBROUTINE LISTING DEFINING CONSTANTS
USED BY THE DIANATEL HARDWARE SWITCH

```
#
#    datmod.inc - include file for DiAnaTel Runtime Link
Library constants
#
#-------------------------------------------------------
------------------------

# Bit masks

     const BLKREQ = 1024;        # add to line number to
cause functions
                         # to block

#-------------------------------- SmartSwitch96 --------
------------------------

# Signal assertion modes

     const SSAS_NONE = 0;        # receive signal bit
looped back to transmit
     const SSAS_ONHOOK = 1;      # send onhook on line
     const SSAS_OFFHOOK = 2;          # send offhook on
line
     const SSAS_PASS = 3;        # pass signal bit
through on connected line

# Signal alert modes

     const SSAL_NO = 0;          # disable signal change
events on line
     const SSAL_YES = 1;         # enable signal change
events on line

# Event types
     const SSEVT_SIGNAL = 01;    # signal bit changed
     const SSEVT_SBOVERFLOW = 02;    # conference bridge
ovrflow
     const SSEVT_FLIPDONE = 03; # flip command is
complete
     const SSEVT_RECVWINK = 04; # inbound wink detected
     const SSEVT_RECVFLASH = 05;     # inbound flash
detected
     const SSEVT_TIMEOUT = 06;  # line's timer expired
```

```
    const MINUDE = 60;          # minimum number for
User Defined Event
                        # maximum is 99

# Configuration Parameters:  Used by Sgetmode() and
Ssetmode()

    const MILLISEC = 0;
    const CLOCK = 1;
    const BOARD_NUMBER = 2;
    const FREQ = 3;
    const FLASHMIN = 4;
    const FLASHMAX = 5;
    const WINKMIN = 6;
    const WINKMAX = 7;
    const CONFERENCES = 8;

#-------------------------------- SmartBridge96 --------
-------------------------
# The following were NOT here in DATMOD.INC, as the VOS
SS96 documentation
# indicates. But these are the values Parity Software
suggested using:

    const SBAT_NONE = 0 ;
    const SBAT_3DB = 1 ;
    const SBAT_6DB = 2 ;

    const SBSU_NONE = 0 ;
    const SBSU_SLIGHT = 1 ;
    const SBSU_MODERATE = 2 ;
    const SBSU_HEAVY = 3 ;

#-------------------------------- EasyAccess24 ---------
-------------------------

# EinPEB(), EoutPEB()

    const EAPEB_PASSTHRU = 0;  # passthrough T1 <-> PEB
    const EAPEB_ONHOOK = 1;        # assert PEB on hook
    const EAPEB_OFFHOOK = 2;   # assert PEB off hook

# Event numbers returned by Egetevt()

    const EAEVT_TNOTERM = 0;   # no term specified
    const EAEVT_TMAXDT = 1;        # maximum DTMF digit
    const EAEVT_TTERMDT = 2;   # terminating DTMF digit
    const EAEVT_TSTOP = 3;     # line now stopped
    const EAEVT_TTIME = 7;     # get tones time out
    const EAEVT_OFFHOOK = 8;   # off hook complete
```

```
    const EAEVT_DIALDONE = 9;   # dial complete
    const EAEVT_ONHOOK = 14;    # on hook complete
    const EAEVT_LC = 20;        # caller hung up
    const EAEVT_RING = 21;      # inbound ringing
    const EAEVT_LCON = 26;      # inbound network now
seized
    const EAEVT_FLIPDONE = 38; # flip done: usually a
wink
    const EAEVT_RECVWINK = 39; # received a wink
    const EAEVT_RECVFLASH = 70;     # received a flash
    const EAEVT_TIMEOUT = 71;   # timer expired
    const EAEVT_T1ALARM = 72;   # t1 now in alarm
    const EAEVT_T1NOALARM = 73;     # t1 out of alarm
    const EAEVT_DCHANNELUP = 74;    # d channl now up
(isdn)
    const EAEVT_DCHANNELDN = 75;    # d channl now down
(isdn)
    const EAEVT_SIGCHANGE  = 76;    # robbed bits
changed


# EgetISDN(), EsetISDN() - service types and allowable
values

    const NSF_SERVICETYPE = 1; # types of service
options
    const NSF_SDN_SERVICE = 225;    #
    const NSF_MEGA800_SERVICE = 226;# AT&T service
    const NSF_MEGACOM_SERVICE = 227;# AT&T service
    const NSF_ACCUNET_SERVICE = 230;# AT&T service
    const NSF_LD_SERVICE = 231;     # AT&T service
    const NSF_INTL800_SERVICE = 232;# AT&T service
    const NSF_MULTIQUEST_SERVICE = 240; # AT&T service

    const NSF_GETCALLERDATA = 2;    # get/set type of
ANI on demand
    const NSF_CPN_PREFERRED = 129;  # this is most
commonly requested
    const NSF_BN_PREFERRED = 130;   # special uses
    const NSF_CPN_ONLY = 131;   # special uses
    const NSF_BN_ONLY = 132;    # special uses
    const NSF_CATSC = 137;      # special uses

    const NSF_VARIABILLTYPE = 3;    # get/set type of
Variabill service
    const NSF_VB_NEWRATE = 16; # set new minute rate
    const NSF_VB_FLATRATE = 17;     # set a flat rate
    const NSF_VB_PCHARGE = 18; # add a premium charge
to call
    const NSF_VB_PCREDIT = 19; # deduct a premium
charge from call
```

```
    const NSF_VB_FREECALL = 24;     # do not charge
caller
    const NSF_WRONGSTATE = 127;     # no call active for
variabill
    const NSF_VB_ACCEPT  = 162;     # variabill charge
was accepted

    const ISDN_SERV_STAT = 4;  # get/put lines in/out
of service
    const ISDN_INSERVICE = 192;     # in service
    const ISDN_OUTSERVICE = 202;    # remove service
    const ISDN_NETMAINTENANCE = 193;# network placed in
maintenance
    const ISDN_NETOUTSERVICE = 194; # network has
removed service

    const ISDN_GETSTATUS = 5;  # get status of isdn
link: up/down


#------------------------------------ TGD24 ------------
-------------------------

# Tgetparm(), Tsetparm() - parameters and allowable
values

    const TG_TONESTYLE = 4;
    const TG_MFDTMFSTYLE = 1;  # style 1 - MF and DTMF
    const TG_DTMFSTYLE = 2;         # style 2 - DTMF and
call progress
    const TG_USERSTYLE = 3;         # style 3 - user
defined tones
```

APPENDIX IX

SUBROUTINE LISTING USED FOR CONFERENCING BY ALL TASKS OTHER THAN THE MAIN PROGRAM

```
# add listener line to conference subroutine
func add_list(scall, aline, logit)

     # check for active call
     if (scall strneq "")

          # defer hang up jump to onsignal
          sc_sigctl("(") ;

          # get specified call's conference number
          conf = glb_get(CALL1CNF+SUBSMAX*scall+subs) ;

          # check for using active specified conference
          if ((conf <= 0) or (conf > CONFMAX))
               voslog("TEST: addlis error - call:
"&scall&" conf: "&conf&" line: "&aline) ;
          else
               # save beginning netline status
               stats = Sstatus(aline);

               # check for inbound netline
               if ((aline >= NETBGN) and (aline <=
NETEND))
                    # add line to specified conference /
leave alerting on
                    code = Saddlis(conf, aline,
SSAS_OFFHOOK, SSAL_YES) ;
               else
                    # add line to specified conference /
leave alerting off
                    code = Saddlis(conf, aline,
SSAS_OFFHOOK, SSAL_NO) ;
               endif
               # check for error return
               if (code <> 0)
                    voslog("Saddlis("&conf&","&aline&")
code "&code&" error") ;
               else
                    if (logit <> 0)
                         voslog("Saddlis("&conf&",
"&aline&"): "&stats&" "&Sstatus(aline)) ;
                    endif
               endif
          endif
```

```
          # allow deferred hang ups
          sc_sigctl(")") ;
     endif
endfunc

# add talker line to conference subroutine
func add_conf(scall, aline, logit)

     # check for active call
     if (scall strneq "")

          # defer hang up jump to onsignal
          sc_sigctl("(") ;

          # get specified call's conference number
          conf = glb_get(CALL1CNF+SUBSMAX*scall+subs) ;

          # check for using active specified conference
          if ((conf <= 0) or (conf > CONFMAX))
               voslog("TEST: addtalk error - call:
"&scall&" conf: "&conf&" line: "&aline) ;
          else
               # save beginning netline status
               stats = Sstatus(aline);

               # check for inbound netline
               if ((aline >= NETBGN) and (aline <=
NETEND))
                    # add line to specified conference /
leave alerting on
                    code = Saddtalk(conf, aline,
SSAS_OFFHOOK, SSAL_YES) ;
               else
                    # add line to specified conference /
leave alerting off
                    code = Saddtalk(conf, aline,
SSAS_OFFHOOK, SSAL_NO) ;
               endif
               # check for error return
               if (code <> 0)
                    voslog("Saddtalk("&conf&","&aline&")
code "&code&" error") ;
               else
                    if (logit <> 0)
                         voslog("Saddtalk("&conf&",
"&aline&"): "&stats&" "&Sstatus(aline)) ;
                    endif
               endif
          endif
```

```
        # allow deferred hang ups
        sc_sigctl(")") ;
    endif
endfunc

# delete listener line from conference subroutine
func del_list(scall, dline, cline, logit)

    # check for active call
    if (scall strneq "")

        # defer hang up jump to onsignal
        sc_sigctl("(") ;

        # get specified call's conference number
        conf = glb_get(CALL1CNF+SUBSMAX*scall+subs) ;

        # check for using active specified conference
        if ((conf <= 0) or (conf > CONFMAX))
            voslog("TEST: dellis error - call:
"&scall&" conf: "&conf&" line: "&dline) ;
        else
            # save beginning netline status
            stats = Sstatus(dline);

            # check for inbound netline
            if ((dline >= NETBGN) and (dline <=
NETEND))
                # remove delete line from specified
conference / leave alerting on
                code = Sdellis(conf, dline,
SSAS_OFFHOOK, SSAL_YES) ;
            else
                # remove delete line from specified
conference / leave alerting off
                code = Sdellis(conf, dline,
SSAS_OFFHOOK, SSAL_NO) ;
            endif
            # check for error return
            if ((code <> 0) and (code <> -24))
                voslog("Sdellis("&conf&","&dline&")
code "&code&" error") ;
            else
                if (logit <> 0)
                    voslog("Sdellis("&conf&",
"&dline&"): "&stats&" "&(stats = Sstatus(dline))) ;
                else
                    stats = Sstatus(dline) ;
                endif
                if ((code = substr(stats, 4, 2)) eq
conf)
```

```
                    voslog("TEST: dellis error -
channel "&dline&" "&stats);
                endif
            endif
        endif
        # check for valid connect line
        if (cline <> 0)
            # check for inbound netline
            if ((dline >= NETBGN) and (dline <=
NETEND))
                # drive delete line receive from
connect line transmit / leave alerting on
                code = Scon(dline, cline, SSAS_PASS,
SSAL_YES) ;
            else
                # drive delete line receive from
connect line transmit / leave alerting off
                code = Scon(dline, cline, SSAS_PASS,
SSAL_NO) ;
            endif
            # check for error return
            if (code <> 0)
                voslog("Scon("&dline&","&cline&") code
"&code&" error") ;
            else
                if (logit <> 0)
                    voslog("Scon("&dline&","&cli
ne&"): "&Sstatus(dline)) ;
                endif
            endif
        endif

        # allow deferred hang ups
        sc_sigctl(")") ;
    endif
endfunc

# delete talker line from conference subroutine
func del_conf(scall, dline, cline, logit)

    # check for active call
    if (scall strneq "")

        # defer hang up jump to onsignal
        sc_sigctl("(") ;

        # get specified call's conference number
        conf = glb_get(CALL1CNF+SUBSMAX*scall+subs) ;

        # check for using active specified conference
        if ((conf <= 0) or (conf > CONFMAX))
```

```
            voslog("TEST: deltalk error - call:
"&scall&" conf: "&conf&" line: "&dline) ;
        else
            # save beginning netline status
            stats = Sstatus(dline);

            # check for inbound netline
            if ((dline >= NETBGN) and (dline <=
NETEND))
                # remove delete line from specified
conference / leave alerting on
                code = Sdeltalk(conf, dline,
SSAS_OFFHOOK, SSAL_YES) ;
            else
                # remove delete line from specified
conference / leave alerting off
                code = Sdeltalk(conf, dline,
SSAS_OFFHOOK, SSAL_NO) ;
            endif
            # check for error return
            if ((code <> 0) and (code <> -24))
                voslog("Sdeltalk("&conf&","&dline&")
code "&code&" error") ;
            else
                if (logit <> 0)
                    voslog("Sdeltalk("&conf&",
"&dline&"): "&stats&" "&(stats = Sstatus(dline))) ;
                else
                    stats = Sstatus(dline) ;
                endif
                if ((code = substr(stats, 4, 2)) eq
conf)
                    voslog("TEST: deltalk error -
channel "&dline&" "&stats);
                endif
            endif
        endif
        # check for valid connect line
        if (cline <> 0)
            # check for inbound netline
            if ((dline >= NETBGN) and (dline <=
NETEND))
                # drive delete line receive from
connect line transmit / leave alerting on
                code = Scon(dline, cline, SSAS_PASS,
SSAL_YES) ;
            else
                # drive delete line receive from
connect line transmit / leave alerting off
                code = Scon(dline, cline, SSAS_PASS,
SSAL_NO) ;
```

```
                endif
                # check for error return
                if (code <> 0)
                     voslog("Scon("&dline&","&cline&") code
"&code&" error") ;
                else
                     if (logit <> 0)
                          voslog("Scon("&dline&","&cli
ne&"): "&Sstatus(dline)) ;
                     endif
                endif
           endif

           # allow deferred hang ups
           sc_sigctl(")") ;
     endif
endfunc

# release unused conference subroutine
func rel_conf(scall)

     # check for active call
     if (scall strneq "")

          # defer hang up jump to onsignal
          sc_sigctl("(") ;

          # get specified call's conference number
          conf = glb_get(CALL1CNF+SUBSMAX*scall+subs) ;

          # check for using active specified conference
          if ((conf > 0) and (conf <= CONFMAX))
               # check if conference still in use
               if   (((scall >= CALLMAX) or
(glb_get(CALL1ACT+SUBSMAX*scall+subs) eq 0))
               and   ((scall >= BARGMAX) or
(glb_get(BARG1SUB+SUBSMAX*scall+subs) eq 0))
               and   ((scall >= FWRDMAX) or
(glb_get(FWRD1SUB+SUBSMAX*scall+subs) eq 0))
               and   ((scall >= FWRDMAX) or
(glb_get(FWRD1ACT+SUBSMAX*scall+subs) eq 0))
               and   ((scall >= MAILMAX) or
(glb_get(MAIL1ACT+SUBSMAX*scall+subs) eq 0))
               and (((scall >= BARGMAX) or
(glb_get(CALL1CNF+SUBSMAX*modulo2(scall)+subs) <> conf))
               or   (((scall >= CALLMAX) or
(glb_get(CALL1ACT+SUBSMAX*modulo2(scall)+subs) eq 0))
               and   ((scall >= BARGMAX) or
(glb_get(BARG1SUB+SUBSMAX*modulo2(scall)+subs) eq 0))
               and   ((scall >= FWRDMAX) or
(glb_get(FWRD1SUB+SUBSMAX*modulo2(scall)+subs) eq 0))
```

```
            and  ((scall >= FWRDMAX) or
(glb_get(FWRD1ACT+SUBSMAX*modulo2(scall)+subs) eq 0))
            and  ((scall >= MAILMAX) or
(glb_get(MAIL1ACT+SUBSMAX*modulo2(scall)+subs) eq 0)))))
                # check to see that conference is
really not in use

                for (code = NETBGN; code <= NETEND;
code++)
                    if (substr((stats =
Sstatus(code)), 4, 2) eq conf)
                        voslog("TEST: software
mistaken about conf being inactive - channel:"&code&"
"&stats);
                        goto log_active ;
                    endif
                endfor
                for (code = ALTBGN; code <= ALTEND;
code++)
                    if (substr((stats =
Sstatus(code)), 4, 2) eq conf)
                        voslog("TEST: software
mistaken about conf being inactive - channel:"&code&"
"&stats);
                        goto log_active ;
                    endif
                endfor
                for (code = GDIDBGN; code <= PAGEEND;
code++)
                    if (substr((stats =
Sstatus(code)), 4, 2) eq conf)
                        voslog("TEST: software
mistaken about conf being inactive - channel:"&code&"
"&stats);
                        goto log_active ;
                    endif
                endfor
                # request conference deallocation
                goto dealloc ;
            else
                # check to see that conference is
really in use

                for (code = NETBGN; code <= NETEND;
code++)
                    if (substr((stats =
Sstatus(code)), 4, 2) eq conf)
                        goto rel_exit ;
                    endif
                endfor
                for (code = ALTBGN; code <= ALTEND;
```

```
code++)
                    if (substr((stats =
Sstatus(code)), 4, 2) eq conf)
                         goto rel_exit ;
                    endif
               endfor
               for (code = GDIDBGN; code <= PAGEEND;
code++)
                    if (substr((stats =
Sstatus(code)), 4, 2) eq conf)
                         goto rel_exit ;
                    endif
               endfor

               voslog("TEST: software mistaken about
conf being active - conf:"&conf&" call:"&scall);
          dealloc:
               # request conference deallocation
               msg_put(mainpid, "E"&conf) ;
               if ((code = msg_get(10)) strneq "E-
OK")
                    voslog("ERROR: code "&code&" when
deallocating conference "&conf) ;
               else
                    voslog("Deallocating conference
"&conf) ;
                    # set specified call's conference
inactive
                    glb_set(CALL1CNF+SUBSMAX*sca
ll+subs, "") ;
               endif
          log_active:
               # log other conferences still active
               for (code = NETBGN; code <= NETEND;
code++)
                    if (substr((stats =
Sstatus(code)), 4, 2) <> 0)
                         voslog("CONF - netline:
"&code&" Sstatus: "&stats) ;
                    endif
               endfor
               for (code = ALTBGN; code <= ALTEND;
code++)
                    if (substr((stats =
Sstatus(code)), 4, 2) <> 0)
                         voslog("CONF - altline:
"&code&" Sstatus: "&stats) ;
                    endif
               endfor
               for (code = GDIDBGN; code <= PAGEEND;
code++)
```

```
                        if (substr((stats =
Sstatus(code)), 4, 2) <> 0)
                            voslog("CONF - vpline:
"&code&" Sstatus: "&stats) ;
                        endif
                    endfor
                endif
            endif
rel_exit:
            # allow deferred hang ups
            sc_sigctl(")") ;
        endif
endfunc
```

APPENDIX X

LISTING OF ASCI TEXT FILES WHICH DEFINE VARIOUS VOS USER SETUP PARAMETERS

```
#    CPB.DEF

#    This file defines the Channel Parameter Block
#    It is used by the MKCPB program to create the
DCB.PAR file.

#    These parameters are set for each telephone line
channel
#    when VOS is started.  The parameters can be changed
when
#    the system is running using the sc_setCPB_word and
#    sc_setCPB_byte functions.

#    This file gives default values to all parameters.

#    See Dialogic documentation for more details
#    on the effect of the parameters.

dtpl_dly = 5  #   Playback DTMF filter time (x10ms)
dt_edge = 2   #   DTMF message edge select 1=lead 2=trail
dtrc_dly = 0  #   Record DTMF delay (x10ms)
sb_size = 200 #   Silence buffer size (bytes)
nbrdna = 2    #   Rings before 'no answer' signalled
stdely = 25   #   Delay after dialling before analysis
(x10ms)
cnosig = 4000 #   No signal timeout delay (x10ms)
lcdly = 10    #   Delay before loop drop = 'connect'
(x10ms)
lcdly1 = 10   #   Delay loop drop to 'connect' signal
(x10ms)
hedge = 1     #   Hello edge to signal connect 1=lead
2=trail
cnosil = 650  #   No silence continuous signal timeout
(x10ms)
lo1tola = 13  #   Accept % below 1st low interval
lo1tolb = 13  #   Accept % below 2nd low interval
lo2tola = 13  #   Accept % above 1st low interval
lo2tolb = 13  #   Accept % above 1st low interval
hi1tola = 13  #   Accept % above high interval
hi1tolb = 13  #   Accept % below high interval
lo1bmax = 90  #   Max time 1st low for busy (x10ms)
lo2bmax = 90  #   Max time 2nd low for busy (x10ms)
```

```
hi1bmax = 90  #  Max time high for busy (x10ms)
nsbusy = 0    #  Nr additional states for busy
log1tch = 15  #  Silence deglitch duration (x10ms)
hig1tch = 19  #  Noise deglitch duration (x10ms)
lo1rmax = 90  #  Max short low double-ring (x10ms)
lo2rmin = 255 #  Min long low double-ring (x10ms)
intflg = 5    #  SIT, PVD options
intfltr = 10  #  Min Intercept tone duration (x10ms)
spdeb = 10    #  Trailing edge silence debounce (x10ms)
hisiz = 90    #  Compare value for prev high (x10ms)
alowmax = 700 #  If prev hi > hisiz use this (x10ms)
blowmax = 530 #  If prev hi < hisiz use this (x10ms)
nbrbeg = 1    #  Nr rings before analysis start
hi1ceil = 78  #  Max 2nd high for retrain (x10ms)
lo1ceil = 58  #  Max 1st low for retrain (x10ms)
lowerfrq = 900    #  Lower accepted freq (Hz)
upperfrq = 1000   #  Upper accepted freq (Hz)
timefrq = 5 .  #  Min Op Intercept signal (x10ms)
rejectfrq = 20    #  Allowed % bad Op Int signal
maxansr = 1000    #  Max duration of hello (x10ms)
ansrdgl = 65535   #  Silence deglitch for answer
pvdmxper = 0     #  (Obsolete)
pvdszwnd = 0     #  (Obsolete)
pvddly = 0       #  (Obsolete)
mxtimefrq = 0    #  Max time for 1st SIT tone (x10ms)
lower2frq = 0    #  Lower bound freq. 2nd SIT tone (Hz)
upper2frq = 0    #  Upper bound freq. 2nd SIT tone (Hz)
time2frq = 0     #  Min time 2nd SIT tone (x10ms)
mxtime2frq = 0   #  Max time 2nd SIT tone (x10ms)
lower3frq = 0    #  Lower bound freq. 3rd SIT tone (Hz)
upper3frq = 0    #  Upper bound freq. 3rd SIT tone (Hz)
time3frq = 0     #  Min time 3rd SIT tone (x10ms)
mxtime3frq = 0   #  Max time 3rd SIT tone (x10ms)
```

```
SS Start                              ;Indicates board type
     Int      = 49                    ;Interrupt number to
install as
     Address = DC00                   ;Location of board
     Millisec = 14                    ;Minimum signaling
duration (140 msec)
     Wink detect                      ;allow wink detection
     Flash detect                     ;allow flash detection
     Clock    = 2                     ;Use PEB 2 (P2) as master
clock
     EVENTMODE = Global               ;all events through
single queue
     CONFERENCES = 6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
;Sixteen 6-way conferences
```

```
SS End                          ;End of SmartSwitch
config data
```

```
#       DCB.DEF

#       This file defines the Dialogic Control Block.
#       It is used by the MKDCB program to create the
DCB.PAR file.

#       These parameters are set once and for all cannot
#       be changed once the system is started.

#       This file gives default values to all
parameters.

#       See Dialogic documentation for more details
#       on the effect of the parameters.

flashchr = &            #  Char in dial string to flash-
hook
flashtm = 50            #  Flash-hook duration (x10ms)
pausetm = 200           #  Dialling pause duration
(x10ms)
digrate = 6053          #  Digitization rate (Hz)
sch_tm = 20             #  Max D40DRV time slice (x50ms)
p_bk = 6                #  Pulse break interval (x10ms)
p_mk = 4                #  Pulse make interval (x10ms)
p_idd = 100             #  Pulse dial inter-digit delay
(x10ms)
t_idd = 5               #  Tone dial inter-digit delay
(x10ms)
oh_dly = 1              #  Off-hook delay (x10ms)
r_on = 3                #  Min ring on interval (x10ms)
r_off = 5               #  Min ring off interval (x10ms)
r_ird = 80              #  Ring count reset delay
(x100ms)
s_bnc = 4               #  Silence debounce interval
(x10ms)
ttdata = 10             #  DTMF tone duration (x10ms)
minpdon = 2             #  Min on interval for pulse
(x10ms)
minpdoff = 2            #  Min off interval for pulse
(x10ms)
minipd = 25             #  Min interdigit time for pulse
(x10ms)
minlcoff = 65535        #  Min loop off duration (x10ms)
                        #  (Default is 65535=-1, which
means "disabled")
```

```
redge = 1                    #  (dcbrfu3) Set to 1 for PEB
environment
maxpdoff = 50                #  Max pulse make duration
(x10ms)
```

```
db_bases = 0          #  Max number of active databases
db_recs = 0           #  Max number of active database
records
db_reclen = 0         #  Max database record length
db_fields = 0         #  Max fields in one database
db_bdes = 0           #  Max active database descriptors
db_rdes = 0           #  Max active record descriptors
msg_size = 32         #  Max length of message string
msg_num = 64          #  Max number of pending messages
glb_size = 16         #  Max length of global variable
glb_num = 350         #  Number of global variables
buf_num = 8           #  Number of 1k buffers
fil_num = 20          #  Max open DOS files
db_ixnum = 0          #  Max number of active index files
db_ixkey = 0          #  Max length of index key
db_ixcache = 0          #  Number of 1k cache buffers per
index file
db_btrieve = 0          #  Btrieve support? 1=YES 0=NO
ser_buff = 0          #  COM port buffer size (bytes)
ser_ports = 0         #  Number of COM ports to support (0
.. 4)
fil_buf = 0           #  fil_copy() buffer size (Kb, or
9999=EMS)
sc_words = 0          #  Max words in sc_playph or
sc_playphm phrase
sc_nshgup = 0         #  Continuous tone treated as hang-up
1=Yes 0=No
arg_size = 3          #  Max chars in spawna/chain argument
fil_locks = 0         #  Max active calls to fil_lock
log_size = 150000     #  Max size of VOS?.LOG
```

```
1 N MAIN
5 N GETDID+CALLIN+CALLOUT+BARGER
4 N FORWARD
2 N VMAIL
2 N PAGE
```

What is claimed is:

1. A communications system comprising:

(a) a central processing unit;

(b) a communications interface for connection to a telephone system and to a signaling system, the signaling system communicating with a plurality of subscribers;

(c) a communications switch connected with said communications interface and with said central processing unit, said central processing unit operating under program control to detect a call from a caller directed to one of said subscribers via said communication interface, to initiate a plurality of essentially simultaneous communications seeking said one of said subscribers in response to said call, at least one of said communications including a signal to said one of said subscribers via said signaling system and at least another one of said communications comprising a telephone call placed via said communication interface and, in response to said one of said subscribers confirming receipt via said communication interface of at least one of said communications, placing said caller and said at least one of said subscribers into direct communication.

2. The communications system of claim 1 wherein said signaling system is a paging system and wherein said subscriber carry pagers which respond to signals

broadcast by said paging system.

3. The communications system of claim 1 wherein said signaling system comprises a computer controlled voice generator and a transducer for reproducing sounds generated by said voice generator, said central processing unit controlling said voice generator, in combination with said transducer, to generate audible signals corresponding to said communications.

4. The communications system of claim 1 wherein said one of said subscribers confirms receipt via said communication interface of at least one of said communications by answering said telephone call.

5. A communications system comprising: a central processing unit and a communications switch connected with the public switched telephone network and with said central processing unit, said central processing unit operating under program control to detect a call from a caller directed to a subscriber via said public switched telephone network and to initiate a plurality of essentially simultaneous communication inquiries via said public switched telephone network seeking said subscriber via different communication paths in said public switched telephone network in response to said call.

6. The communications system of claim 5 wherein said plurality of essentially simultaneous communication inquiries comprise a plurality telephone calls initiated to a plurality of different telephone numbers.

7. The communications system of claim 6 wherein at

least one of said plurality of telephone calls continues to ring irrespective of another one of said plurality of telephone calls being answered.

8. A method of placing a caller into telecommunication with a subscriber, the method comprising the steps of:

(i) detecting receipt of a call from said caller directed to said subscriber;

(ii) in response to receipt of said call, initiating a page to said subscriber and at the same time initiating a forward leg call to a stored telephone address;

(iii) placing said forward leg call in at least half duplex telecommunication with said caller, if answered; and

(iv) detecting receipt of a call from said subscriber, and, in response thereto, placing said subscriber in at least half duplex telecommunication with said caller.

9. The method of placing a caller into telecommunication with a subscriber as claimed in claim 8, and while in telecommunication with the first mentioned caller, detecting receipt of a new call from a new caller directed to said subscriber, sending a signal to said subscriber indicating that said new call has been detected and responsive to a command entered by said subscriber, toggling said subscriber between the first mentioned caller and the new caller.

10. The method of placing a caller into telecommunication with a subscriber as claimed in claim

8, and when said caller, said subscriber and a party on said forward leg are in telecommunication with each other, sensing a command entered by said subscriber to disconnect said party and disconnecting said party in response thereto.

11. The method of placing a caller into telecommunication with a subscriber as claimed in claim 8, and when said caller, said subscriber and a party on said forward leg are all in telecommunication with each other, allowing said party and said caller to remain in telecommunication with each other in spite of said subscriber disconnecting.

12. The method of placing a caller into telecommunication with a subscriber as claimed in claim 8, and wherein at least two of (i) said caller, (ii) said subscriber and (iii) a party on said forward leg are in telecommunication with each other, sensing a command entered by at least one of said two to add a further forward leg to said telecommunication.

13. The method of placing a caller into telecommunication with a subscriber as claimed in claim 12, wherein the sensing step senses a command entered by any party in the telecommunication to add a further forward leg to said telecommunication.

14. The method of placing a caller into telecommunication with a subscriber as claimed in claim 12, wherein said further forward leg is a connection made to a voice mail system.

15. The method of placing a caller into

telecommunication with a subscriber as claimed in claim 12, wherein said further forward leg is an outbound telephone call made to a third party.

16. The method of placing a caller into telecommunication with a subscriber as claimed in claim 8, and wherein at least two of (i) said caller, (ii) said subscriber and (iii) a party on said forward leg are in telecommunication with each other, sensing a command entered by at least one of said two to record a voice memo and recording said voice memo in response to said command.

17. The method of placing a caller into telecommunication with a subscriber as claimed in claim 8, wherein said subscriber is initially placed in half duplex communication with said caller and thereafter, in response to a command entered by said subscriber, the communication between the subscriber and the caller is changed to full duplex communication.

18. The method of placing a caller into telecommunication with a subscriber as claimed in claim 8, wherein a party answering a forward leg call may hang up before the subscriber is placed into communication with the caller without disconnecting the caller.

19. The method of placing a caller into telecommunication with a subscriber as claimed in claim 18, wherein the caller is connected to a voice mail system in response to a command entered by the caller.

20. The method of placing a caller into telecommunication with a subscriber as claimed in claim

18, wherein the caller is connected to a voice mail system after waiting for the subscriber for a period of time.

21. The method of placing a caller into telecommunication with a subscriber as claimed in claim 19 or 20, wherein when the caller is connected to voice mail the subscriber may listen to the caller's communications with voice mail in a listen mode wherein said subscriber cannot be heard by said caller.

22. The method of placing a caller into telecommunication with a subscriber as claimed in claim 8, wherein the caller is connected to a voice mail system in response to a command entered by the caller.

23. The method of claim 8, further including the steps of sensing a message from a telephone system advising that a telephone connected therewith has been powered up and in response thereto changing said telephone address.

24. The method of claim 8, further including the steps of sensing a message from a telephone system advising that a telephone connected therewith has been powered up and in response thereto telephoning said subscriber at predetermined telephone address to deliver a predetermined courtesy message advising them of that fact.

25. The method of claim 8, wherein said stored telephone address is stored in a database along with a schedule and wherein, in step (ii), said stored telephone address is selected from said database in accordance with said schedule.

26. The method of claim 8, wherein multiple forward leg calls are initiated to different stored telephone addresses simultaneously.

27. The method of claim 8, wherein in step (ii) multiple pages are initiated concurrently to different locations.

28. The method of claim 27, wherein one of said different locations is a paging system and wherein another of said locations is a speaker whereat an audible page is delivered.

29. A method of placing a caller into telecommunication with a subscriber, the method comprising the steps of:

(i) detecting receipt of a call from said caller directed to said subscriber;

(ii) in response to receipt of said call, initiating a forward leg call to a stored telephone address;

(iii) connecting said call with a voice mail facility;

(iv) connecting a person answering said forward leg call in half duplex telecommunication with said caller while said caller is connected to said voice mail facility; and

(v) detecting receipt of a command from said person and, in response thereto, placing said person in full duplex telecommunication with said caller.

30. The method of placing a caller into telecommunication with a subscriber as claimed in claim 29, further including the steps of:

(vi) initiating a page to a subscriber;

(vii) sensing receipt of a call from said subscriber in response to said page, and placing said subscriber in half duplex telecommunication with said caller; and

(viii) detecting receipt of a command from said subscriber and, in response thereto, placing said subscriber in full duplex telecommunication with said caller.

31. The method of placing a caller into telecommunication with a subscriber as claimed in claims 29 or 30, and when said subscriber, said caller, said person on said forward leg are in telecommunication with each other, sensing a command entered by said subscriber to disconnect said person and disconnecting said party in response thereto.

32. The method of placing a caller into telecommunication with a subscriber as claimed in claims 29 or 30, and when at least two of (i) said caller, (ii) said subscriber and (iii) a party on said forward leg are in telecommunication with each other, sensing a command entered by at least one of said two to add a further forward leg to said telecommunication and adding said further forward leg in response to said command.

33. The method of placing a caller into telecommunication with a subscriber as claimed in claim 32, wherein said further forward leg is an outbound telephone call made to a third party.

34. The method of placing a caller into telecommunication with a subscriber as claimed in claims 29 or 30, and wherein at least two of (i) said caller, (ii) said subscriber and (iii) a party on said forward leg are in telecommunication with each other, sensing a command entered by at least one of said two to record a voice memo and recording said voice memo in response to said command.

35. The method of placing a caller into telecommunication with a subscriber as claimed in claims 29 or 30, wherein said person is initially placed in half duplex communication with said caller and thereafter, in response to a command entered by said person, the communication between the person and the caller is changed to full duplex communication.

36. A method of placing a caller into telecommunication with a subscriber, the method comprising the steps of:

(i) detecting receipt of a call from said caller directed to said subscriber;

(ii) in response to receipt of said call, initiating a page to a subscriber;

(iii) connecting said caller with a voice mail facility;

(iv) sensing receipt of a call from said subscriber and placing said subscriber in half duplex telecommunication with said caller while said caller is connected to said voice mail facility; and

(v) detecting receipt of a command from said subscriber and, in response thereto, placing said subscriber in full duplex telecommunication with said caller.

37. The method of claim 36 further including the steps of:

(vi) initiating at least one forward leg call in response to receipt of the call from the caller; and

(vii) wherein step (iii) occurs if at least one forward leg call is not promptly answered or at least after receipt of the call from the subscriber if at least one forward leg call has not previously answered.

38. A method of placing a caller into telecommunication with a subscriber, the method comprising the steps of:

(i) detecting receipt of a call from said caller directed to said subscriber;

(ii) in response to receipt of said call, initiating a forward leg call to a stored telephone address and initiating a page to a subscriber;

(iii) connecting a person answering said forward leg call telecommunication with said caller;

(iv) sensing receipt of a call from said subscriber and placing said subscriber in half duplex telecommunication with said caller while said subscriber is in telecommunication with said person; and

(v) detecting receipt of a command from said subscriber and, in response thereto, placing said subscriber in full duplex telecommunication with said caller and said person.

39. A communications system comprising:

(a) a central processing unit;

(b) a communications interface for connection to a telephone system and to a signaling system, the signaling system including a central transmitter which communicates with a plurality of receiving units carried by subscribers;

(c) a communications switch connected with said communications interface and with said central processing unit, said central processing unit operating under program control to detect a call from a caller directed to one of said subscribers via said communication interface, to initiate a plurality of essentially simultaneous communication inquiries via said interface seeking said one of said subscribers via different communication paths in response to said call,

at least one of said inquiries including a signal to said one of said subscribers via said signaling system and placing said caller and said at least one of said subscribers into communication after said one of said subscribers responds to said communication inquires.

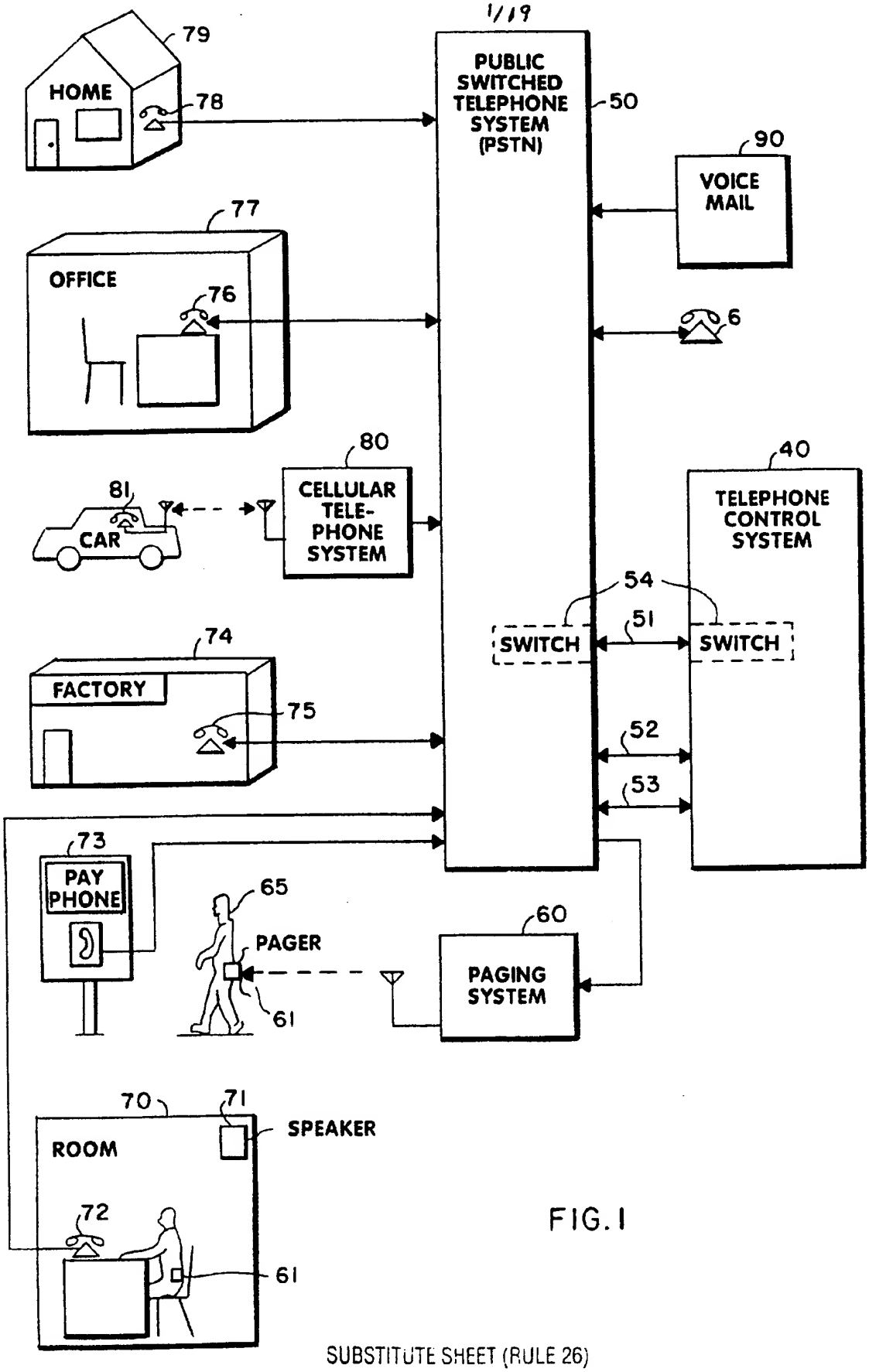

79
HOME
78
PUBLIC SWITCHED TELEPHONE SYSTEM (PSTN)
50
90
VOICE MAIL
77
OFFICE
76
6
80
81
CAR
CELLULAR TELE-PHONE SYSTEM
40
TELEPHONE CONTROL SYSTEM
54
SWITCH
51
SWITCH
74
FACTORY
75
52
53
73
PAY PHONE
65
PAGER
60
PAGING SYSTEM
61
70
71
SPEAKER
ROOM
72
61

FIG. 1

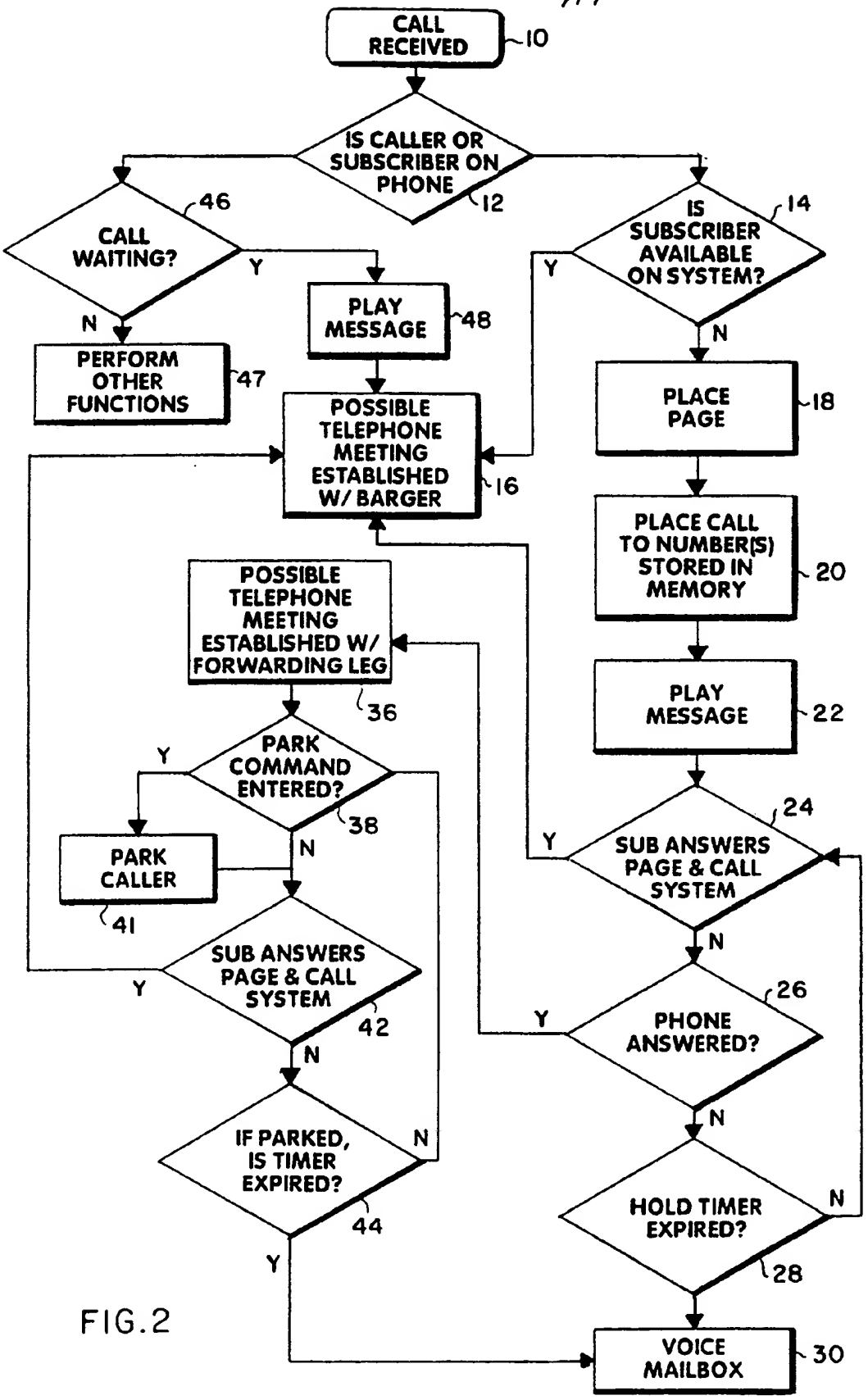

CALL RECEIVED
10
IS CALLER OR SUBSCRIBER ON PHONE
12
CALL WAITING?
46
Y
N
PLAY MESSAGE
48
PERFORM OTHER FUNCTIONS
47
IS SUBSCRIBER AVAILABLE ON SYSTEM?
14
Y
N
PLACE PAGE
18
POSSIBLE TELEPHONE MEETING ESTABLISHED W/ BARGER
16
PLACE CALL TO NUMBER(S) STORED IN MEMORY
20
POSSIBLE TELEPHONE MEETING ESTABLISHED W/ FORWARDING LEG
36
PLAY MESSAGE
22
PARK COMMAND ENTERED?
38
Y
N
PARK CALLER
41
SUB ANSWERS PAGE & CALL SYSTEM
24
Y
N
SUB ANSWERS PAGE & CALL SYSTEM
42
Y
N
PHONE ANSWERED?
26
Y
N
IF PARKED, IS TIMER EXPIRED?
44
N
Y
HOLD TIMER EXPIRED?
28
N
VOICE MAILBOX
30

FIG.2

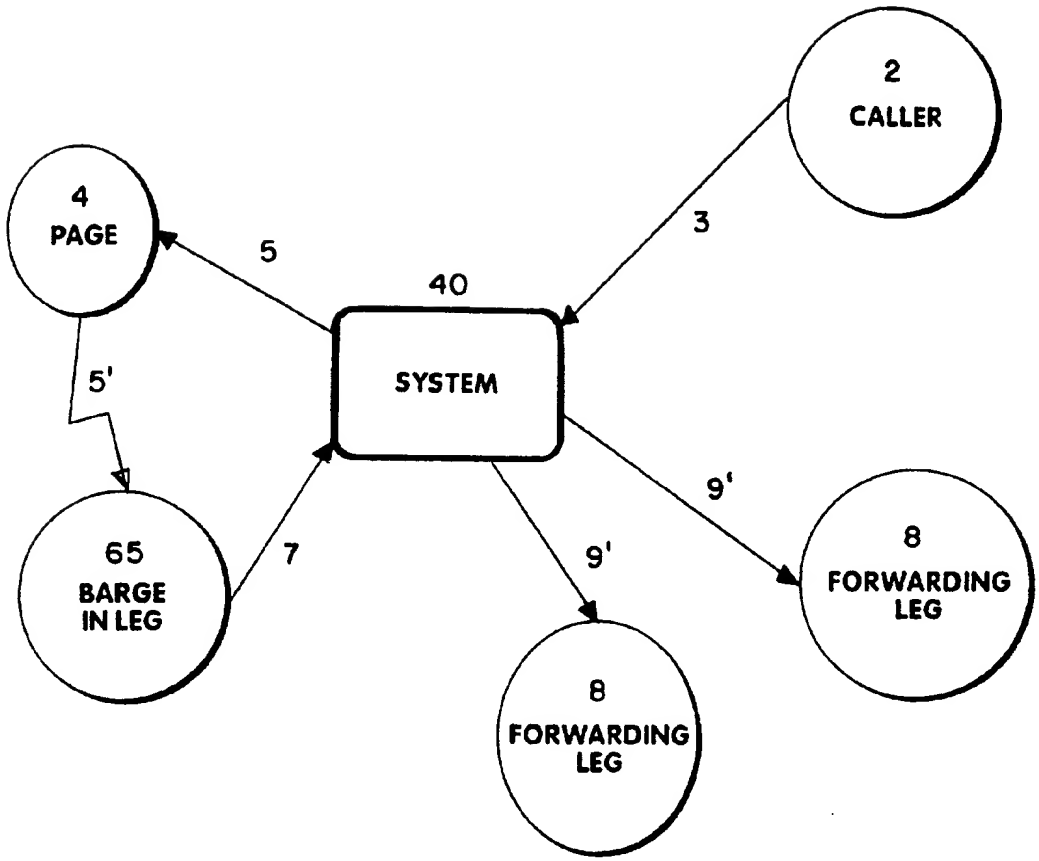

2
CALLER
4
PAGE
5
3
40
SYSTEM
5'
9'
65
BARGE
IN LEG
7
9'
8
FORWARDING
LEG
8
FORWARDING
LEG

FIG.3

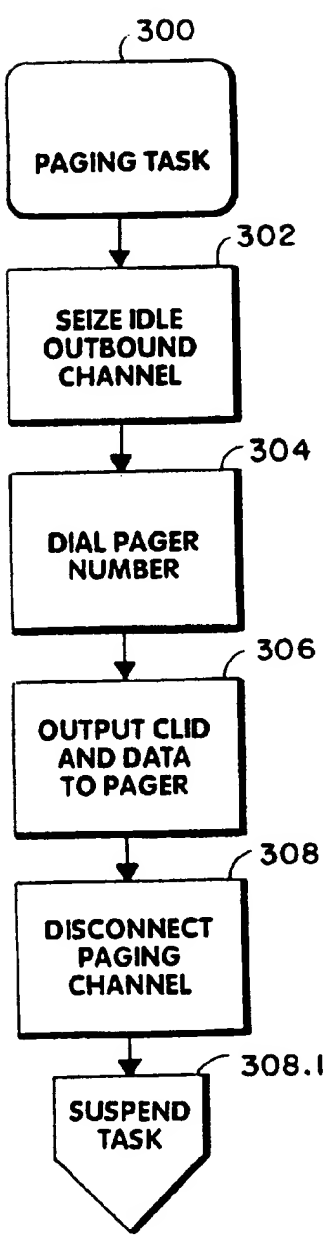


FIG.4c

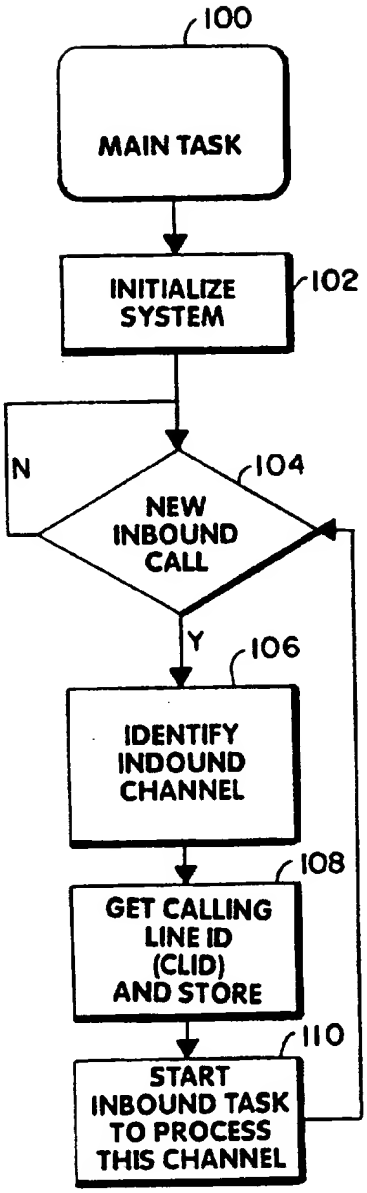


FIG.4a

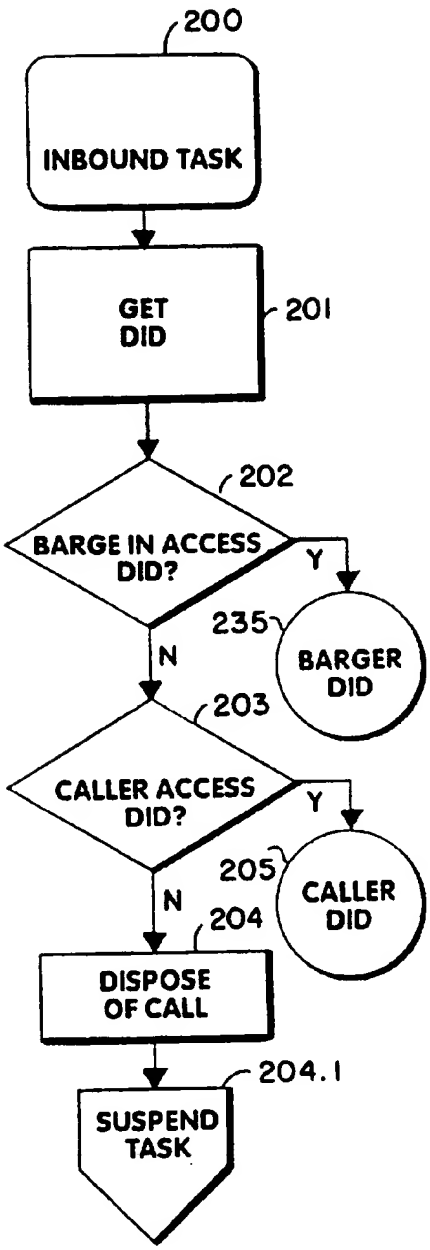


FIG.4b

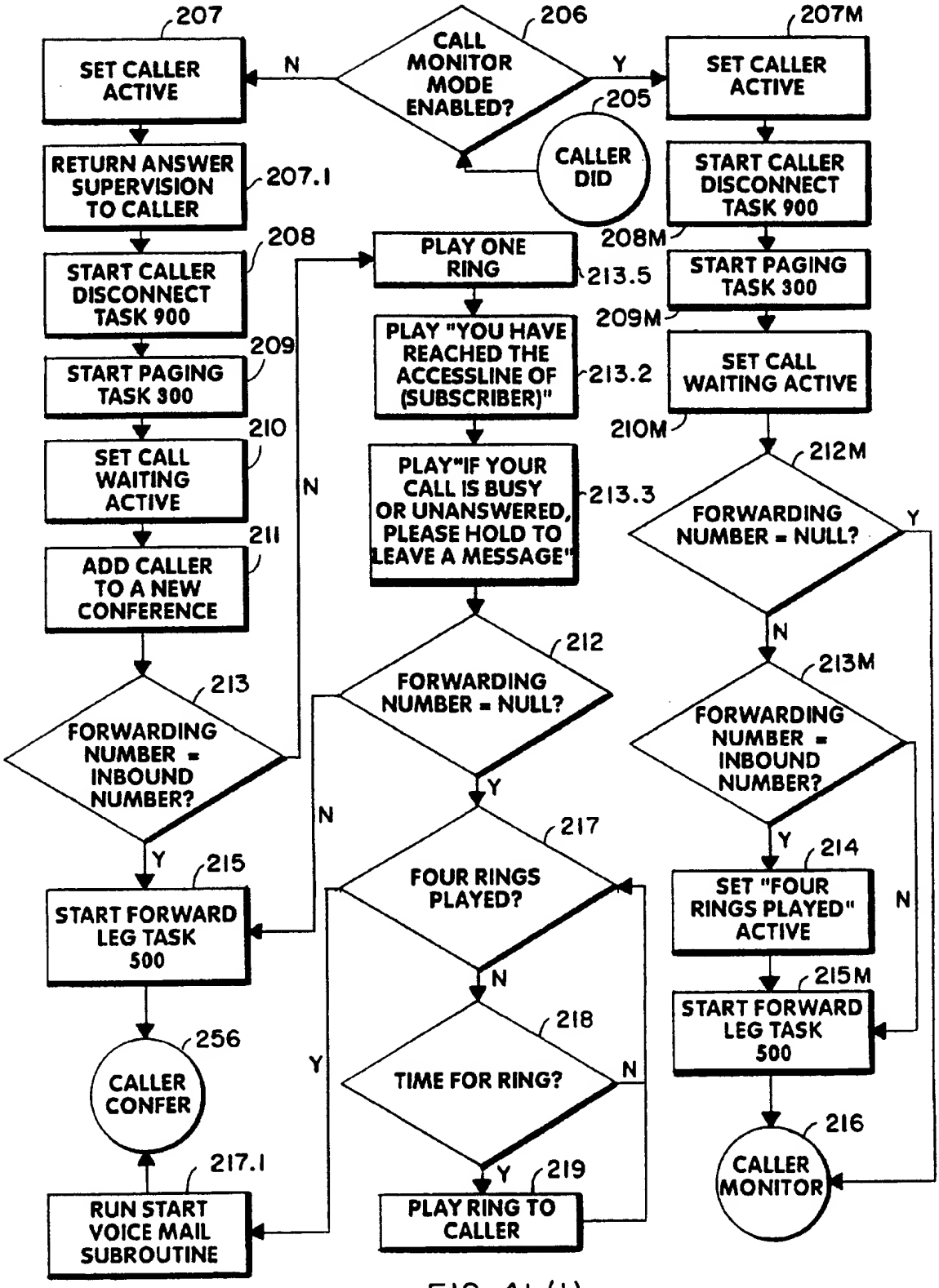

207
SET CALLER ACTIVE
206
CALL MONITOR MODE ENABLED?
N
Y
207M
SET CALLER ACTIVE
205
CALLER DID
RETURN ANSWER SUPERVISION TO CALLER
207.1
START CALLER DISCONNECT TASK 900
208M
208
START CALLER DISCONNECT TASK 900
PLAY ONE RING
213.5
START PAGING TASK 300
209M
209
START PAGING TASK 300
PLAY "YOU HAVE REACHED THE ACCESSLINE OF (SUBSCRIBER)"
213.2
SET CALL WAITING ACTIVE
210M
210
SET CALL WAITING ACTIVE
PLAY"IF YOUR CALL IS BUSY OR UNANSWERED, PLEASE HOLD TO LEAVE A MESSAGE"
213.3
212M
FORWARDING NUMBER = NULL?
Y
N
211
ADD CALLER TO A NEW CONFERENCE
212
FORWARDING NUMBER = NULL?
213M
FORWARDING NUMBER = INBOUND NUMBER?
213
FORWARDING NUMBER = INBOUND NUMBER?
N
Y
Y
N
217
FOUR RINGS PLAYED?
214
SET "FOUR RINGS PLAYED" ACTIVE
N
215
START FORWARD LEG TASK 500
N
215M
START FORWARD LEG TASK 500
218
TIME FOR RING?
256
CALLER CONFER
Y
N
216
CALLER MONITOR
217.1
RUN START VOICE MAIL SUBROUTINE
Y
219
PLAY RING TO CALLER

FIG. 4b(1)

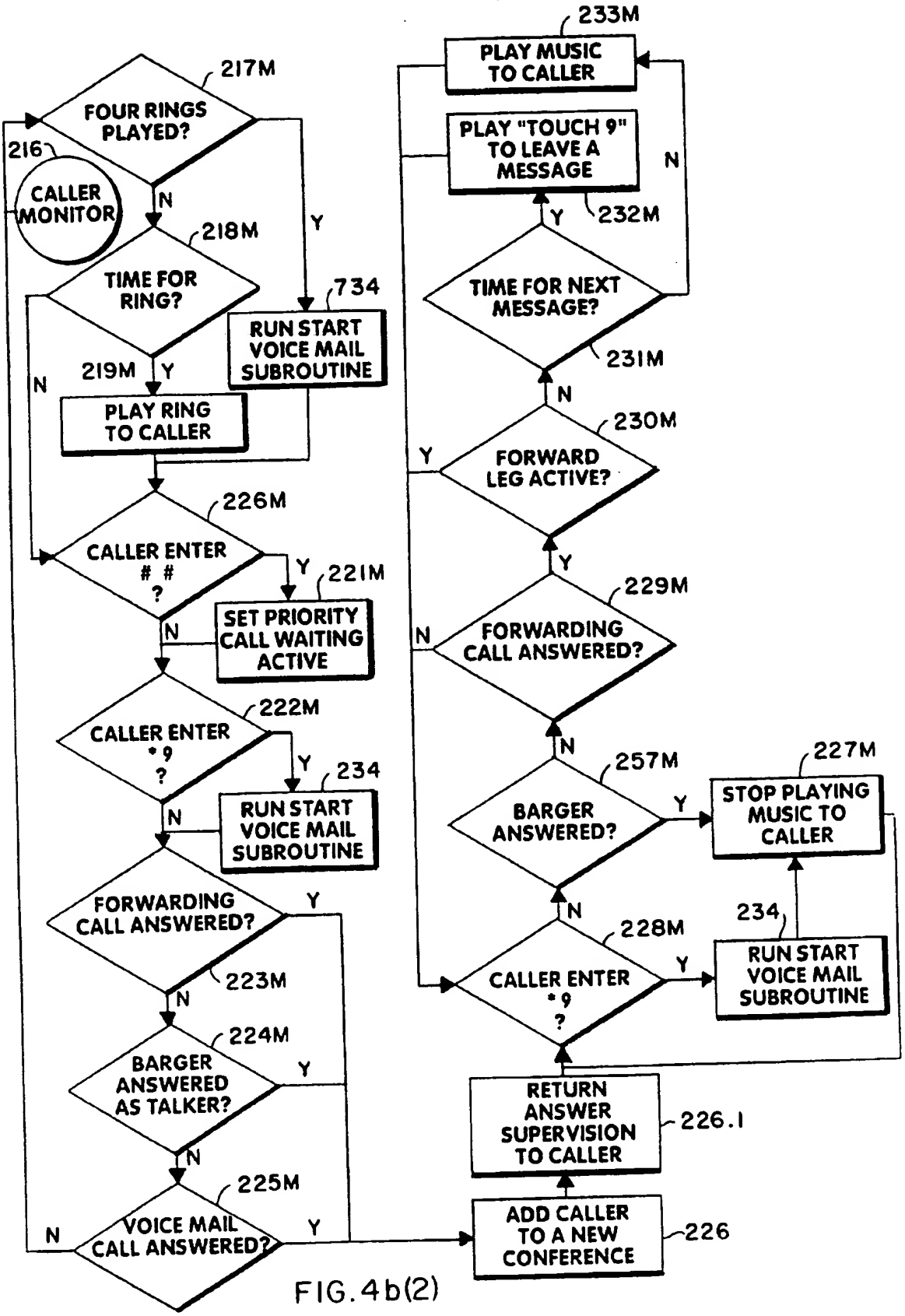

217M
FOUR RINGS PLAYED?
216
CALLER MONITOR
N
Y
218M
TIME FOR RING?
734
RUN START VOICE MAIL SUBROUTINE
219M
Y
N
PLAY RING TO CALLER
226M
CALLER ENTER # # ?
Y
221M
SET PRIORITY CALL WAITING ACTIVE
N
222M
CALLER ENTER * 9 ?
Y
234
RUN START VOICE MAIL SUBROUTINE
N
FORWARDING CALL ANSWERED?
Y
223M
N
224M
BARGER ANSWERED AS TALKER?
Y
N
225M
VOICE MAIL CALL ANSWERED?
N
Y
233M
PLAY MUSIC TO CALLER
PLAY "TOUCH 9" TO LEAVE A MESSAGE
N
Y
232M
TIME FOR NEXT MESSAGE?
231M
N
230M
FORWARD LEG ACTIVE?
Y
Y
229M
FORWARDING CALL ANSWERED?
N
N
257M
BARGER ANSWERED?
Y
227M
STOP PLAYING MUSIC TO CALLER
N
228M
CALLER ENTER * 9 ?
Y
234
RUN START VOICE MAIL SUBROUTINE
RETURN ANSWER SUPERVISION TO CALLER
226.1
ADD CALLER TO A NEW CONFERENCE
226

FIG. 4b(2)

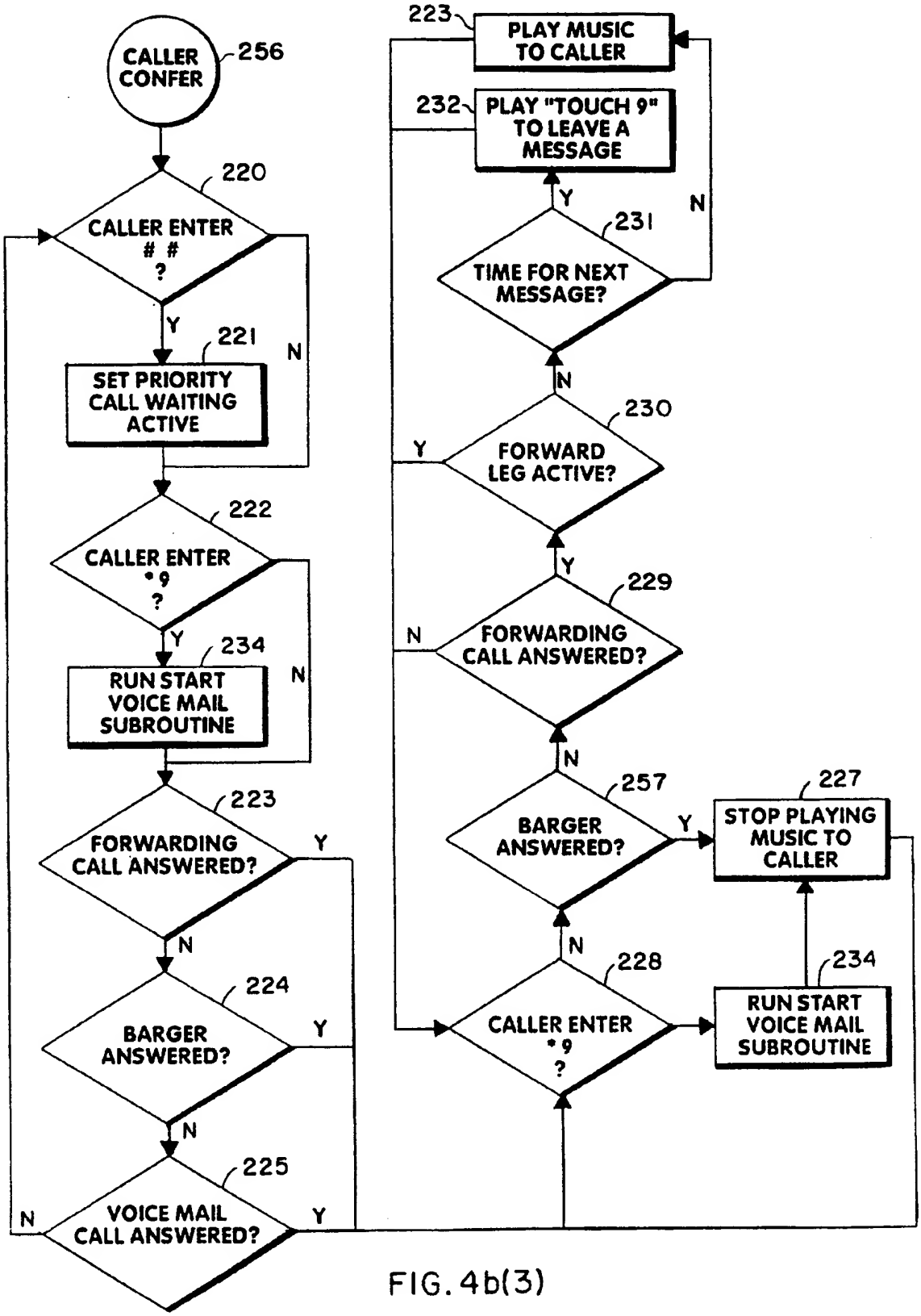

CALLER CONFER
256
220
CALLER ENTER # # ?
Y
N
221
SET PRIORITY CALL WAITING ACTIVE
222
CALLER ENTER *9 ?
Y
N
234
RUN START VOICE MAIL SUBROUTINE
223
FORWARDING CALL ANSWERED?
Y
N
224
BARGER ANSWERED?
Y
N
225
VOICE MAIL CALL ANSWERED?
N
Y
223
PLAY MUSIC TO CALLER
232
PLAY "TOUCH 9" TO LEAVE A MESSAGE
Y
N
231
TIME FOR NEXT MESSAGE?
N
230
FORWARD LEG ACTIVE?
Y
Y
229
FORWARDING CALL ANSWERED?
N
N
257
BARGER ANSWERED?
Y
227
STOP PLAYING MUSIC TO CALLER
N
228
CALLER ENTER *9 ?
234
RUN START VOICE MAIL SUBROUTINE

FIG. 4b(3)

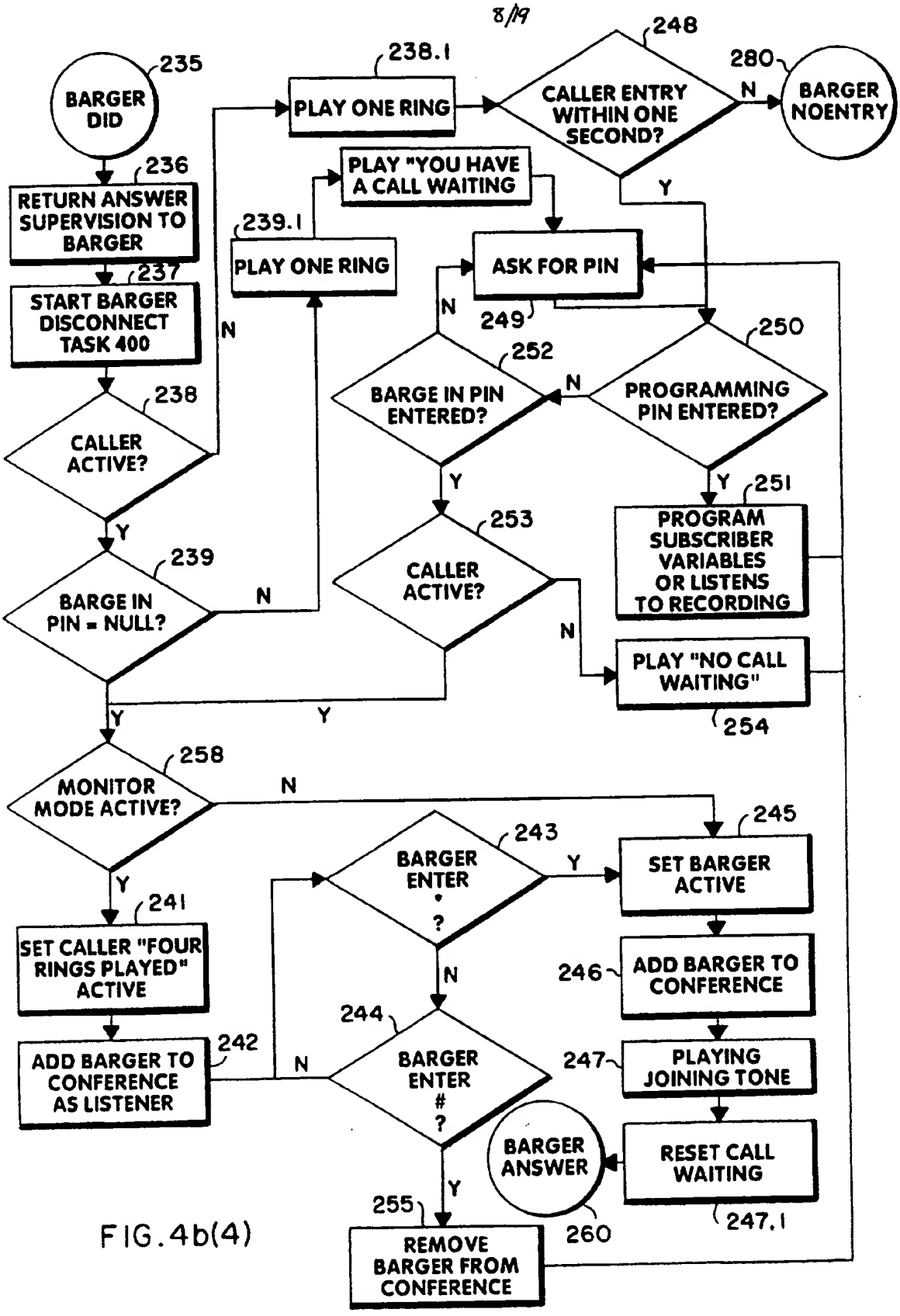

235
BARGER DID
236
RETURN ANSWER SUPERVISION TO BARGER
237
START BARGER DISCONNECT TASK 400
238
CALLER ACTIVE?
N
Y
239
BARGE IN PIN = NULL?
N
Y
238.1
PLAY ONE RING
239.1
PLAY ONE RING
PLAY "YOU HAVE A CALL WAITING
248
CALLER ENTRY WITHIN ONE SECOND?
N
280
BARGER NOENTRY
Y
ASK FOR PIN
N
249
250
PROGRAMMING PIN ENTERED?
N
252
BARGE IN PIN ENTERED?
Y
Y
251
PROGRAM SUBSCRIBER VARIABLES OR LISTENS TO RECORDING
253
CALLER ACTIVE?
N
PLAY "NO CALL WAITING"
254
Y
Y
258
MONITOR MODE ACTIVE?
N
Y
241
SET CALLER "FOUR RINGS PLAYED" ACTIVE
242
ADD BARGER TO CONFERENCE AS LISTENER
N
244
BARGER ENTER # ?
N
243
BARGER ENTER * ?
Y
245
SET BARGER ACTIVE
246
ADD BARGER TO CONFERENCE
247
PLAYING JOINING TONE
247.1
RESET CALL WAITING
BARGER ANSWER
260
Y
255
REMOVE BARGER FROM CONFERENCE

FIG.4b(4)

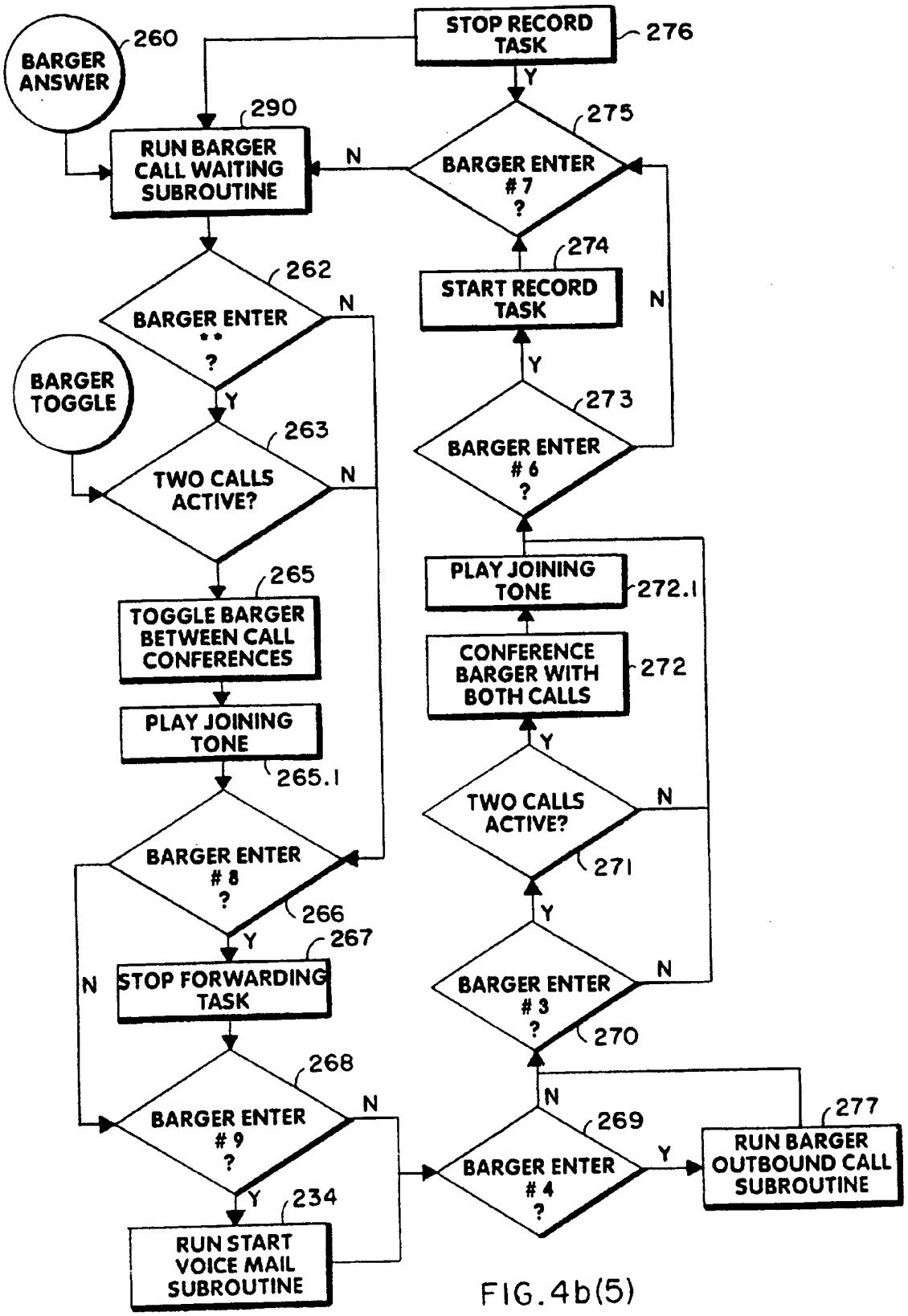

BARGER ANSWER
260
RUN BARGER CALL WAITING SUBROUTINE
290
BARGER ENTER ** ?
262
BARGER TOGGLE
TWO CALLS ACTIVE?
263
TOGGLE BARGER BETWEEN CALL CONFERENCES
265
PLAY JOINING TONE
265.1
BARGER ENTER # 8 ?
266
STOP FORWARDING TASK
267
BARGER ENTER # 9 ?
268
RUN START VOICE MAIL SUBROUTINE
234
BARGER ENTER # 4 ?
269
BARGER ENTER # 3 ?
270
TWO CALLS ACTIVE?
271
CONFERENCE BARGER WITH BOTH CALLS
272
PLAY JOINING TONE
272.1
BARGER ENTER # 6 ?
273
START RECORD TASK
274
BARGER ENTER # 7 ?
275
STOP RECORD TASK
276
RUN BARGER OUTBOUND CALL SUBROUTINE
277
Y
N

FIG. 4b(5)

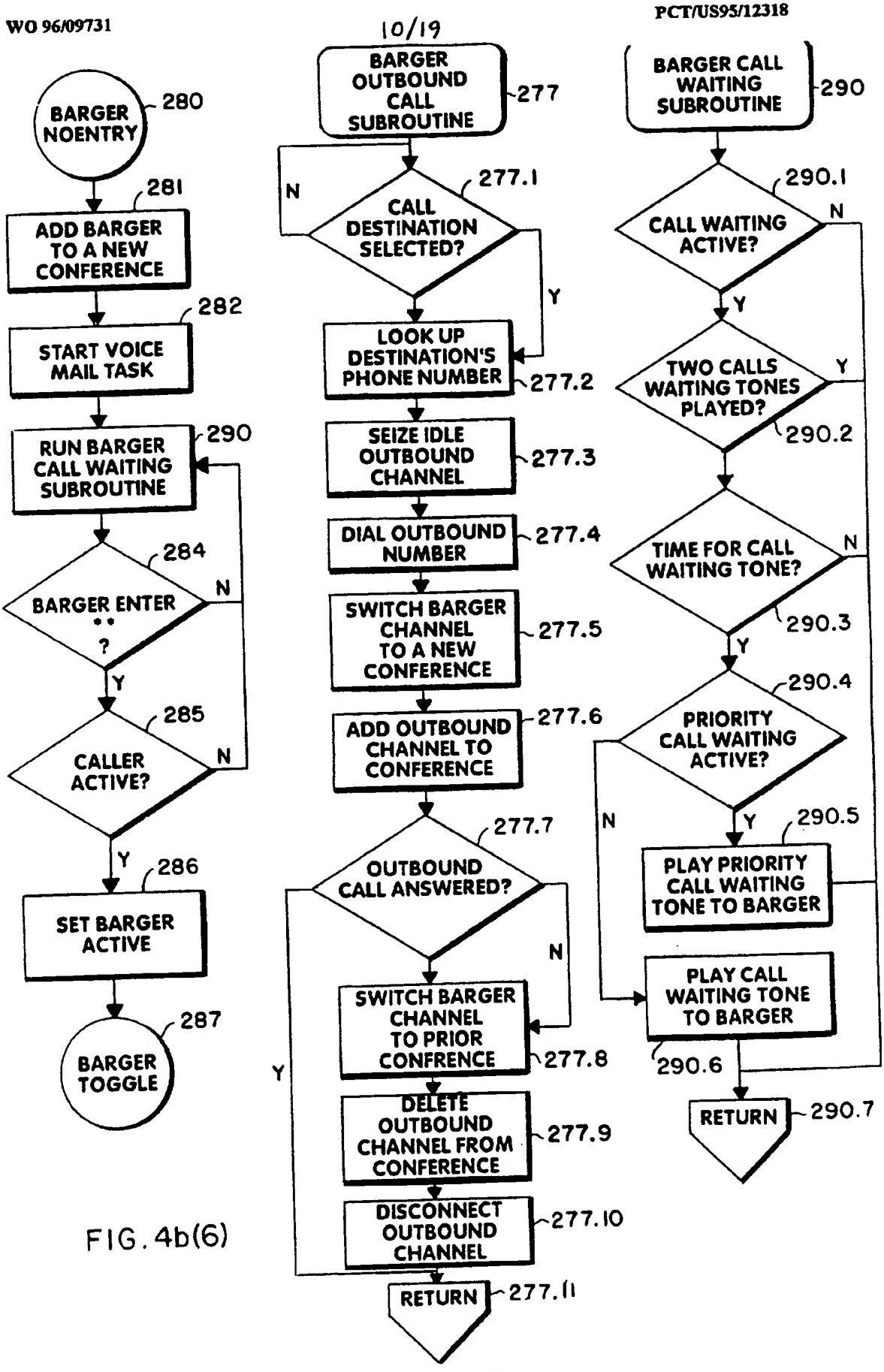

BARGER NOENTRY
280
ADD BARGER TO A NEW CONFERENCE
281
START VOICE MAIL TASK
282
RUN BARGER CALL WAITING SUBROUTINE
290
BARGER ENTER **?
284
N
Y
CALLER ACTIVE?
285
N
Y
SET BARGER ACTIVE
286
BARGER TOGGLE
287
BARGER OUTBOUND CALL SUBROUTINE
277
N
CALL DESTINATION SELECTED?
277.1
Y
LOOK UP DESTINATION'S PHONE NUMBER
277.2
SEIZE IDLE OUTBOUND CHANNEL
277.3
DIAL OUTBOUND NUMBER
277.4
SWITCH BARGER CHANNEL TO A NEW CONFERENCE
277.5
ADD OUTBOUND CHANNEL TO CONFERENCE
277.6
OUTBOUND CALL ANSWERED?
277.7
N
Y
SWITCH BARGER CHANNEL TO PRIOR CONFRENCE
277.8
DELETE OUTBOUND CHANNEL FROM CONFERENCE
277.9
DISCONNECT OUTBOUND CHANNEL
277.10
RETURN
277.11
BARGER CALL WAITING SUBROUTINE
290
CALL WAITING ACTIVE?
290.1
N
Y
TWO CALLS WAITING TONES PLAYED?
290.2
Y
TIME FOR CALL WAITING TONE?
290.3
N
Y
PRIORITY CALL WAITING ACTIVE?
290.4
N
Y
PLAY PRIORITY CALL WAITING TONE TO BARGER
290.5
PLAY CALL WAITING TONE TO BARGER
290.6
RETURN
290.7

FIG. 4b(6)

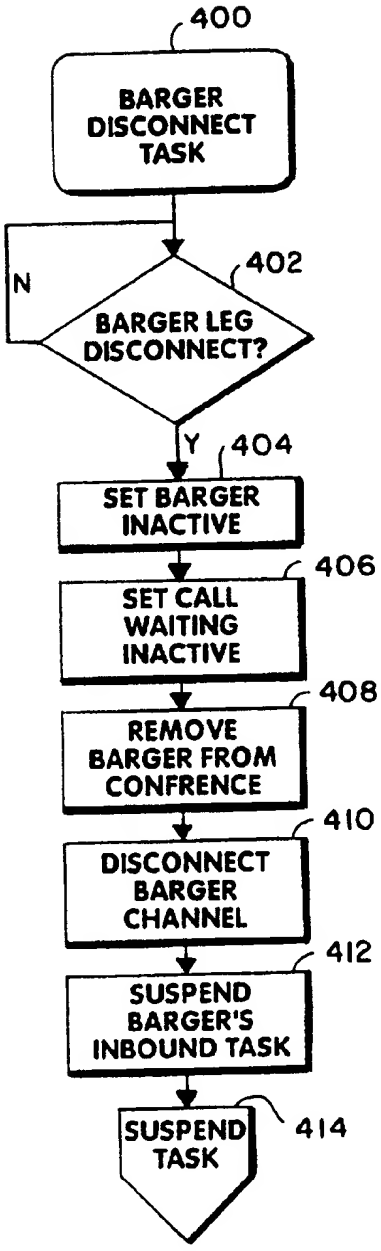


FIG. 4d

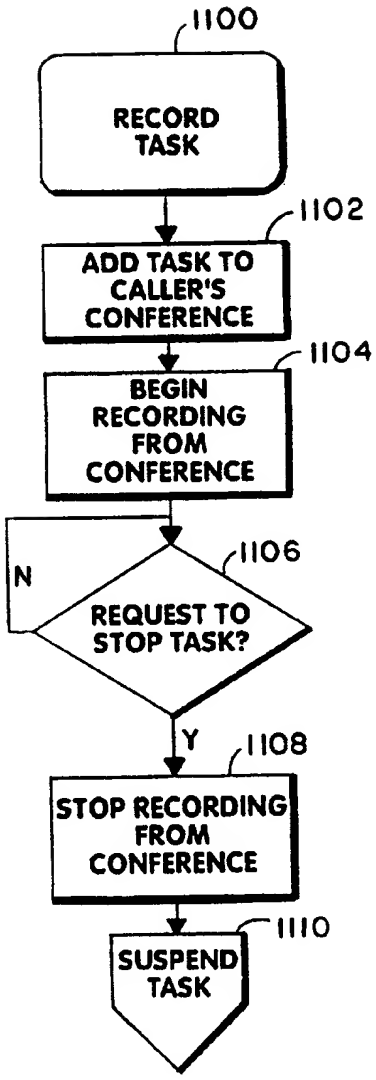


FIG. 4k

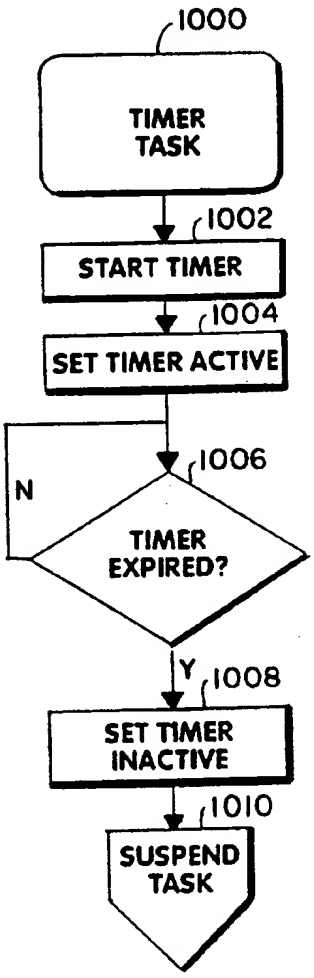


FIG. 4j

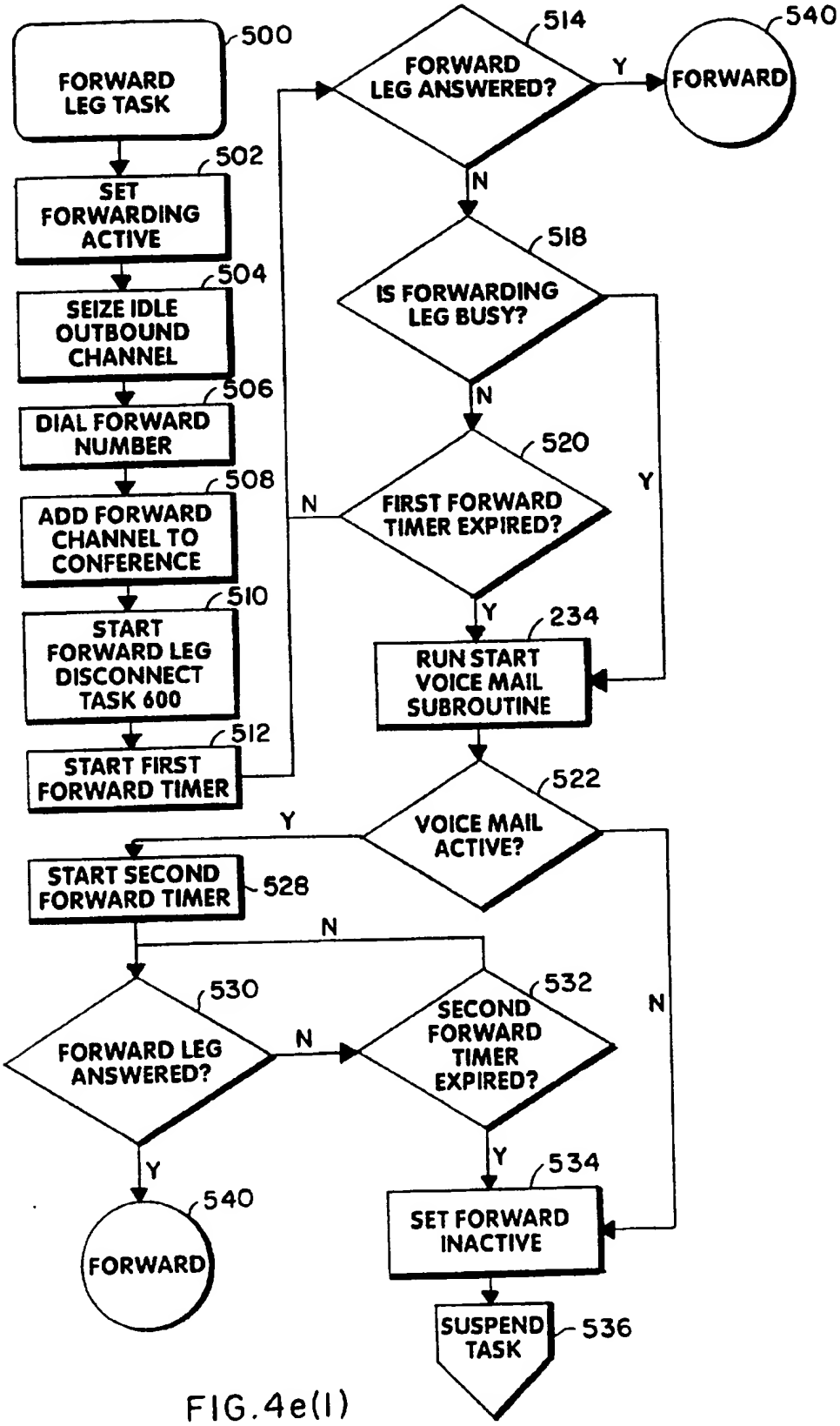

FORWARD LEG TASK
500
SET FORWARDING ACTIVE
502
SEIZE IDLE OUTBOUND CHANNEL
504
DIAL FORWARD NUMBER
506
ADD FORWARD CHANNEL TO CONFERENCE
508
START FORWARD LEG DISCONNECT TASK 600
510
START FIRST FORWARD TIMER
512
FORWARD LEG ANSWERED?
514
Y
FORWARD
540
N
IS FORWARDING LEG BUSY?
518
N
Y
FIRST FORWARD TIMER EXPIRED?
520
N
Y
RUN START VOICE MAIL SUBROUTINE
234
VOICE MAIL ACTIVE?
522
Y
N
START SECOND FORWARD TIMER
528
FORWARD LEG ANSWERED?
530
N
SECOND FORWARD TIMER EXPIRED?
532
N
Y
FORWARD
540
Y
SET FORWARD INACTIVE
534
SUSPEND TASK
536

FIG. 4e(1)

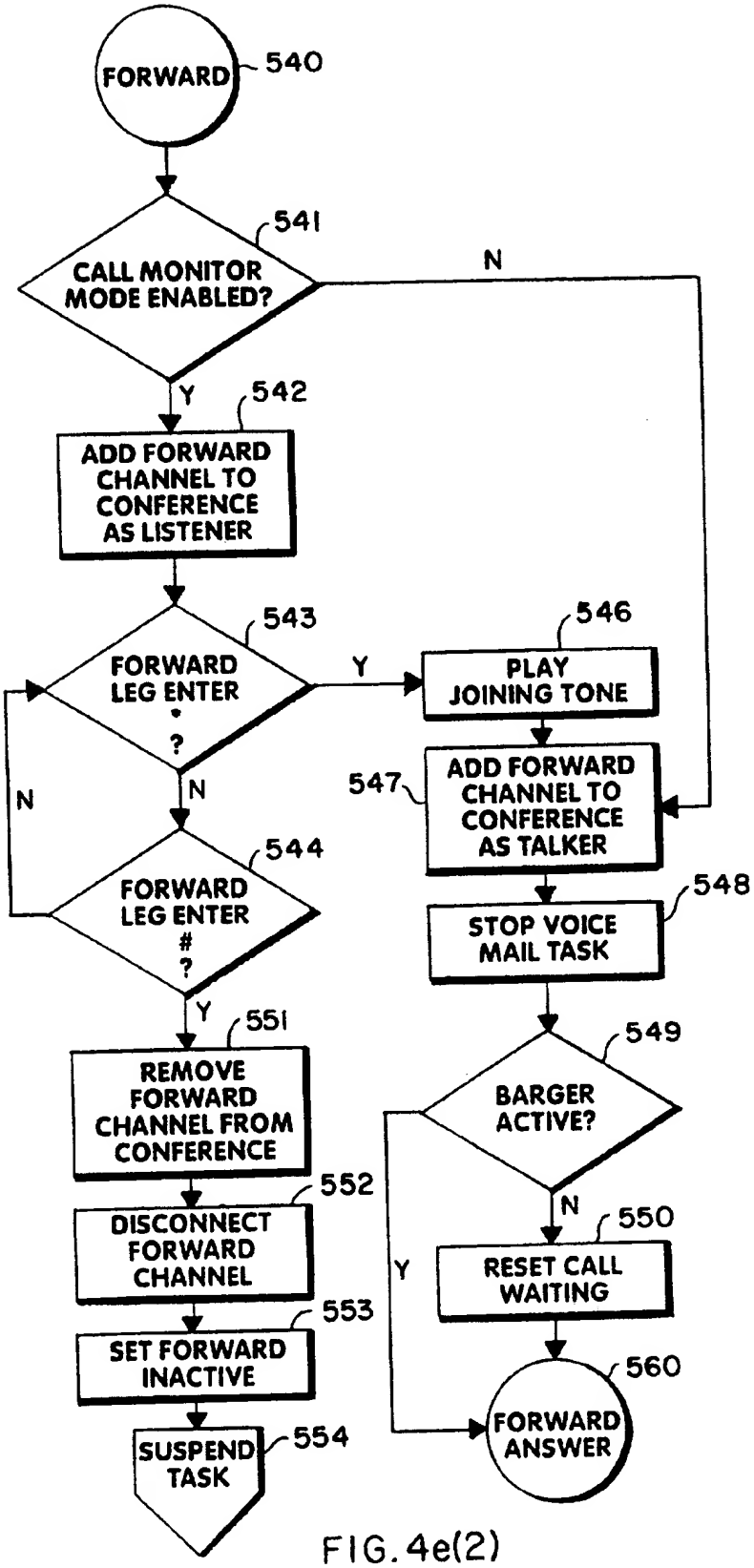

FORWARD
540
541
CALL MONITOR MODE ENABLED?
N
Y
542
ADD FORWARD CHANNEL TO CONFERENCE AS LISTENER
543
FORWARD LEG ENTER * ?
Y
546
PLAY JOINING TONE
N
N
547
ADD FORWARD CHANNEL TO CONFERENCE AS TALKER
544
FORWARD LEG ENTER # ?
548
STOP VOICE MAIL TASK
Y
551
REMOVE FORWARD CHANNEL FROM CONFERENCE
549
BARGER ACTIVE?
552
DISCONNECT FORWARD CHANNEL
N
550
RESET CALL WAITING
Y
553
SET FORWARD INACTIVE
560
FORWARD ANSWER
554
SUSPEND TASK

FIG. 4e(2)

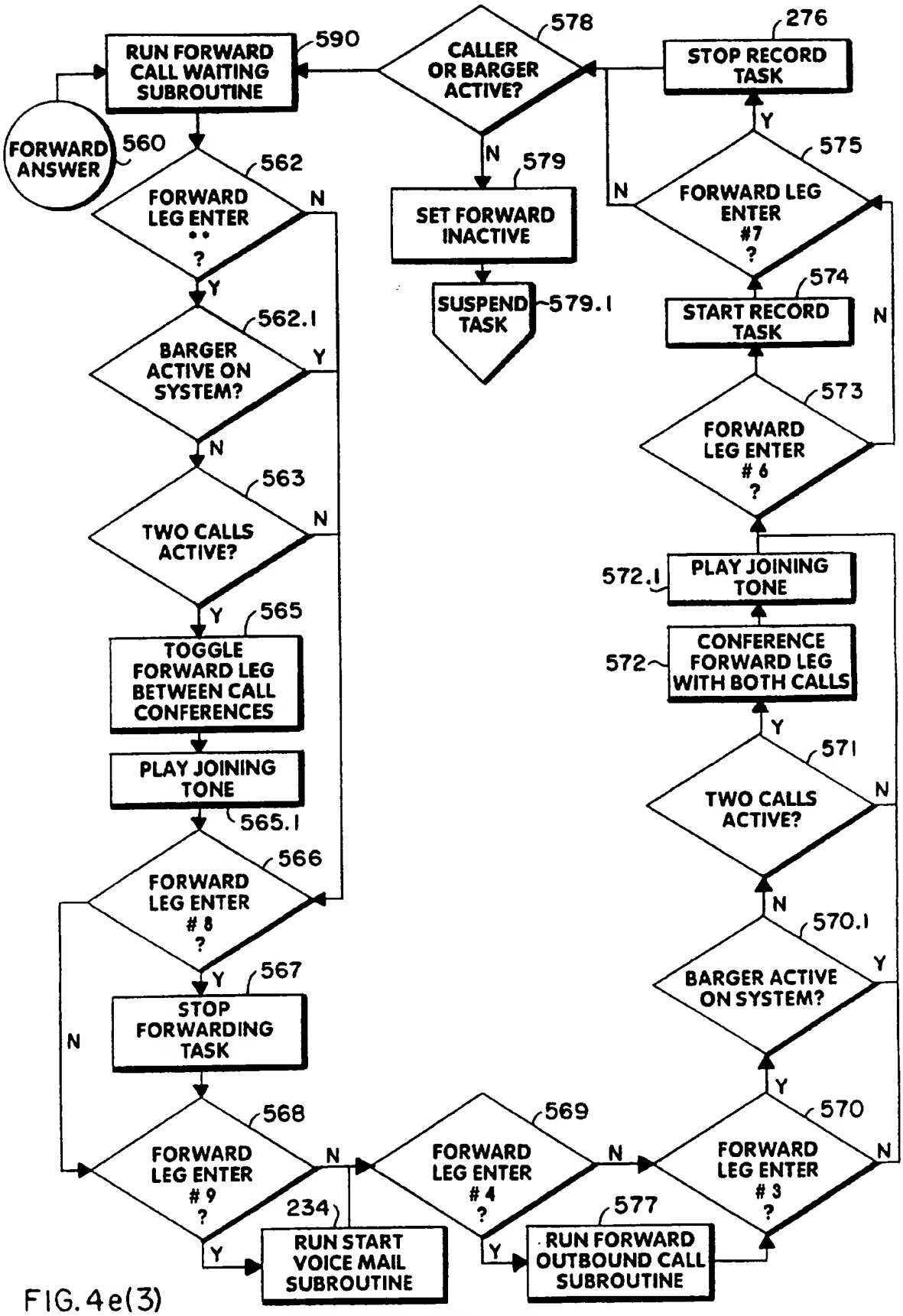

RUN FORWARD CALL WAITING SUBROUTINE
590
CALLER OR BARGER ACTIVE?
578
STOP RECORD TASK
276
FORWARD ANSWER
560
FORWARD LEG ENTER ** ?
562
SET FORWARD INACTIVE
579
FORWARD LEG ENTER #7 ?
575
SUSPEND TASK
579.1
START RECORD TASK
574
BARGER ACTIVE ON SYSTEM?
562.1
FORWARD LEG ENTER # 6 ?
573
TWO CALLS ACTIVE?
563
PLAY JOINING TONE
572.1
TOGGLE FORWARD LEG BETWEEN CALL CONFERENCES
565
CONFERENCE FORWARD LEG WITH BOTH CALLS
572
PLAY JOINING TONE
565.1
TWO CALLS ACTIVE?
571
FORWARD LEG ENTER # 8 ?
566
BARGER ACTIVE ON SYSTEM?
570.1
STOP FORWARDING TASK
567
FORWARD LEG ENTER # 9 ?
568
FORWARD LEG ENTER # 4 ?
569
FORWARD LEG ENTER # 3 ?
570
234
RUN START VOICE MAIL SUBROUTINE
577
RUN FORWARD OUTBOUND CALL SUBROUTINE
Y
N

FIG. 4e(3)

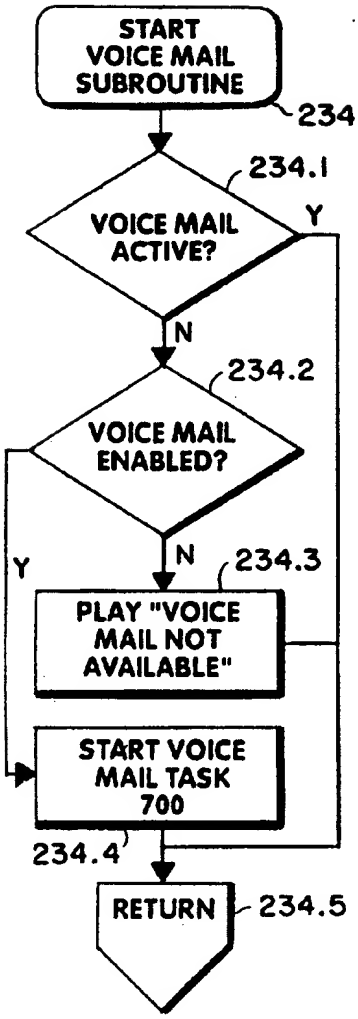
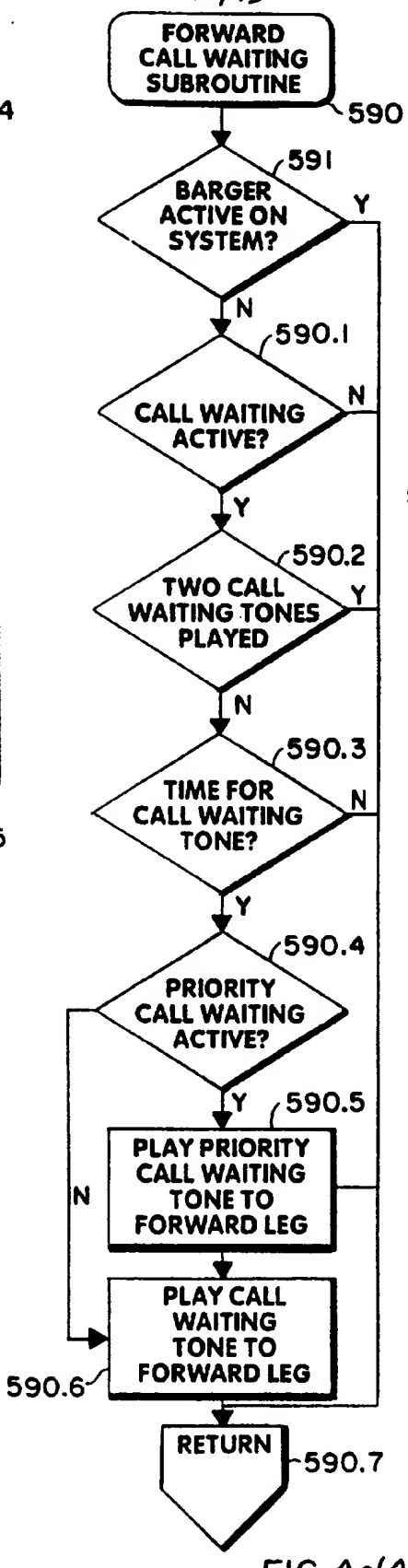


FIG.4b(7)



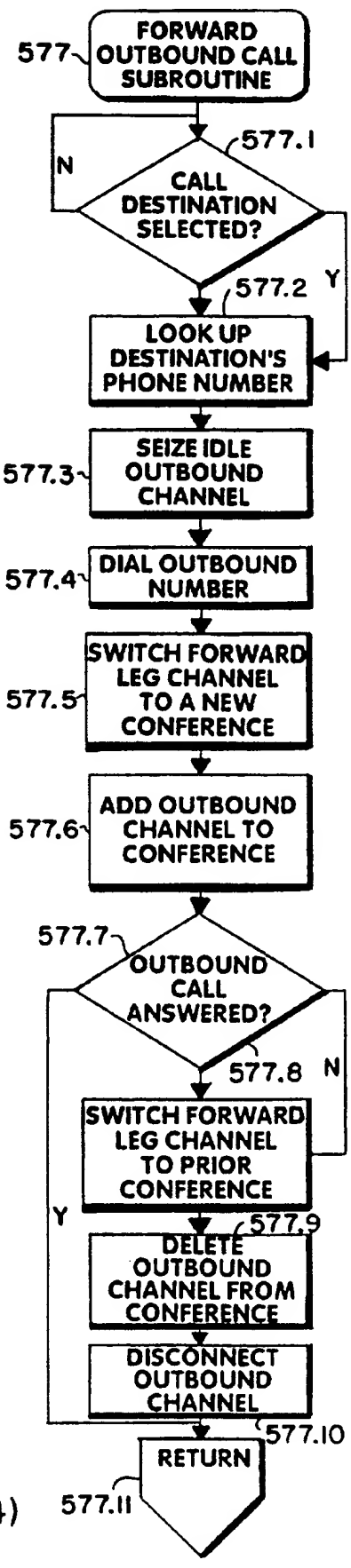


FIG.4e(4)

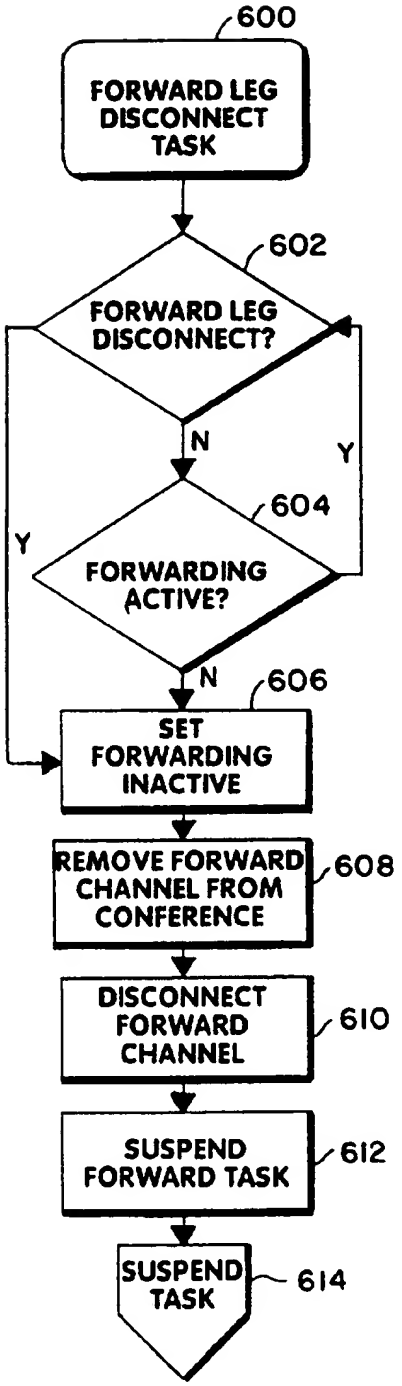


FIG. 4f

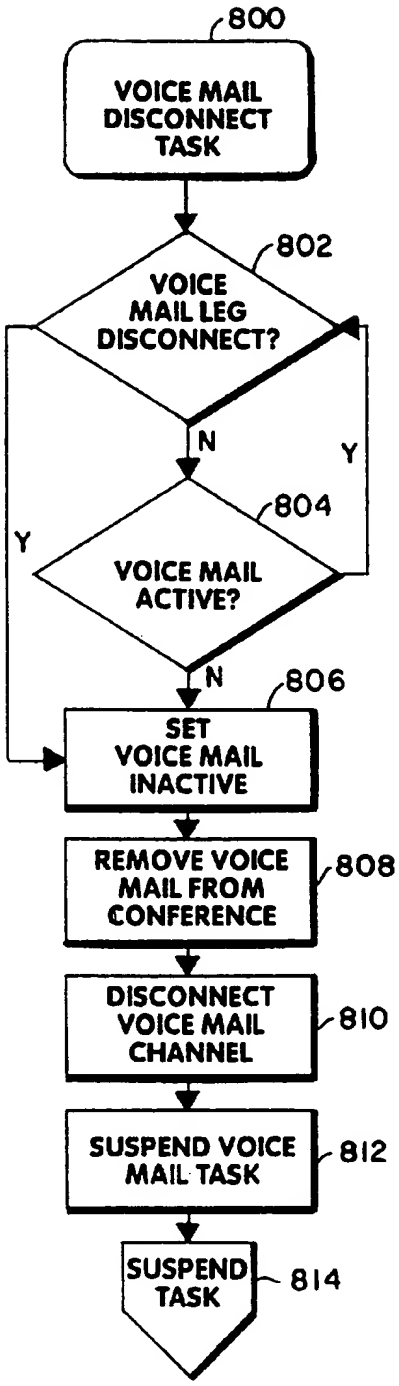


FIG. 4h

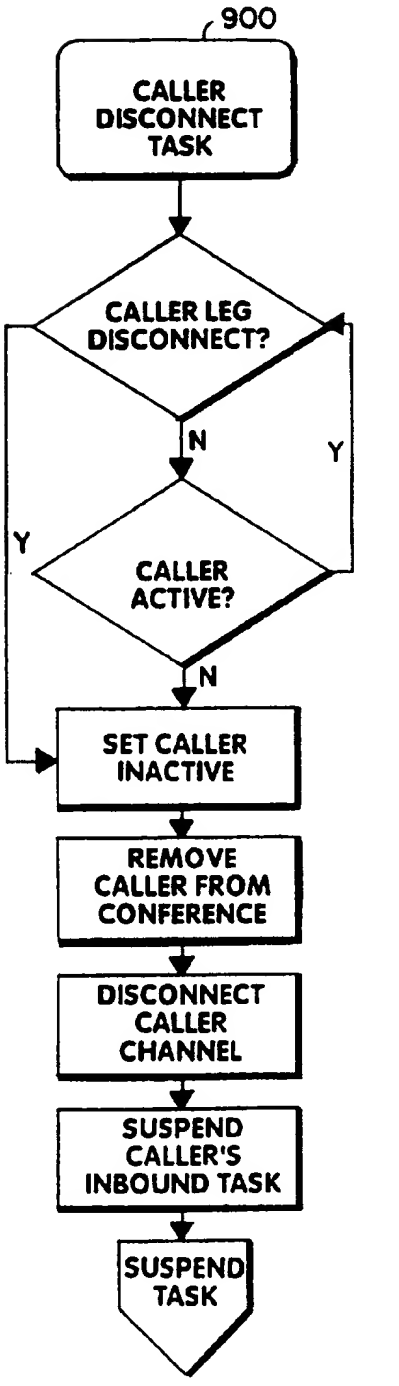


FIG. 4i

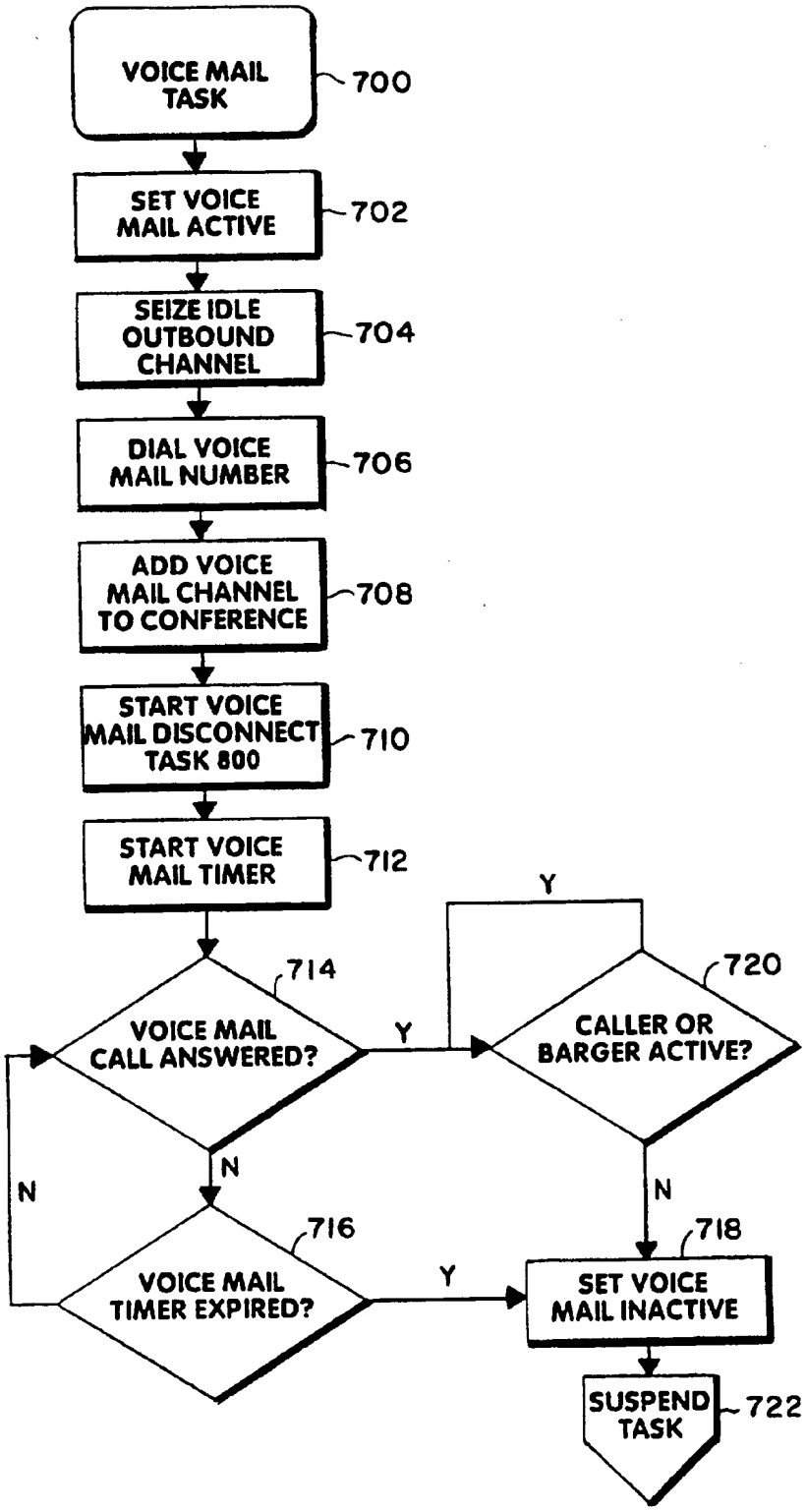

VOICE MAIL TASK
700
SET VOICE MAIL ACTIVE
702
SEIZE IDLE OUTBOUND CHANNEL
704
DIAL VOICE MAIL NUMBER
706
ADD VOICE MAIL CHANNEL TO CONFERENCE
708
START VOICE MAIL DISCONNECT TASK 800
710
START VOICE MAIL TIMER
712
VOICE MAIL CALL ANSWERED?
714
Y
CALLER OR BARGER ACTIVE?
720
N
N
VOICE MAIL TIMER EXPIRED?
716
Y
SET VOICE MAIL INACTIVE
718
SUSPEND TASK
722

FIG. 4g

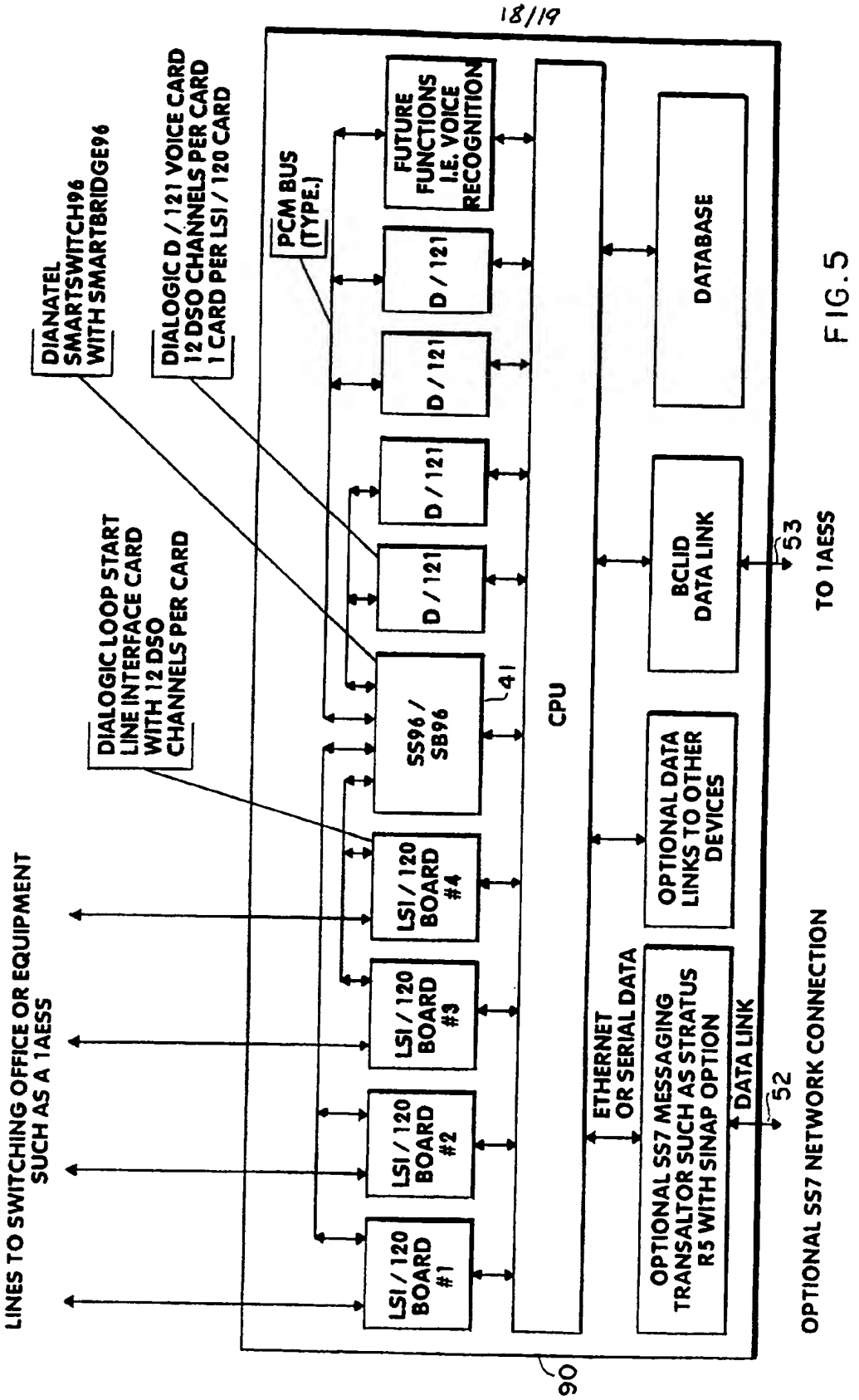

DIANATEL SMARTSWITCH96 WITH SMARTBRIDGE96
DIALOGIC D / 121 VOICE CARD 12 DSO CHANNELS PER CARD 1 CARD PER LSI / 120 CARD
DIALOGIC LOOP START LINE INTERFACE CARD WITH 12 DSO CHANNELS PER CARD
LINES TO SWITCHING OFFICE OR EQUIPMENT SUCH AS A 1AESS
PCM BUS (TYPE.)
FUTURE FUNCTIONS I.E. VOICE RECOGNITION
D / 121
D / 121
D / 121
D / 121
SS96 / SB96
41
LSI / 120 BOARD #4
LSI / 120 BOARD #3
LSI / 120 BOARD #2
LSI / 120 BOARD #1
CPU
DATABASE
BCLID DATA LINK
53
TO 1AESS
OPTIONAL DATA LINKS TO OTHER DEVICES
ETHERNET OR SERIAL DATA
OPTIONAL SS7 MESSAGING TRANSALTOR SUCH AS STRATUS R5 WITH SINAP OPTION
DATA LINK
52
OPTIONAL SS7 NETWORK CONNECTION
90

FIG. 5

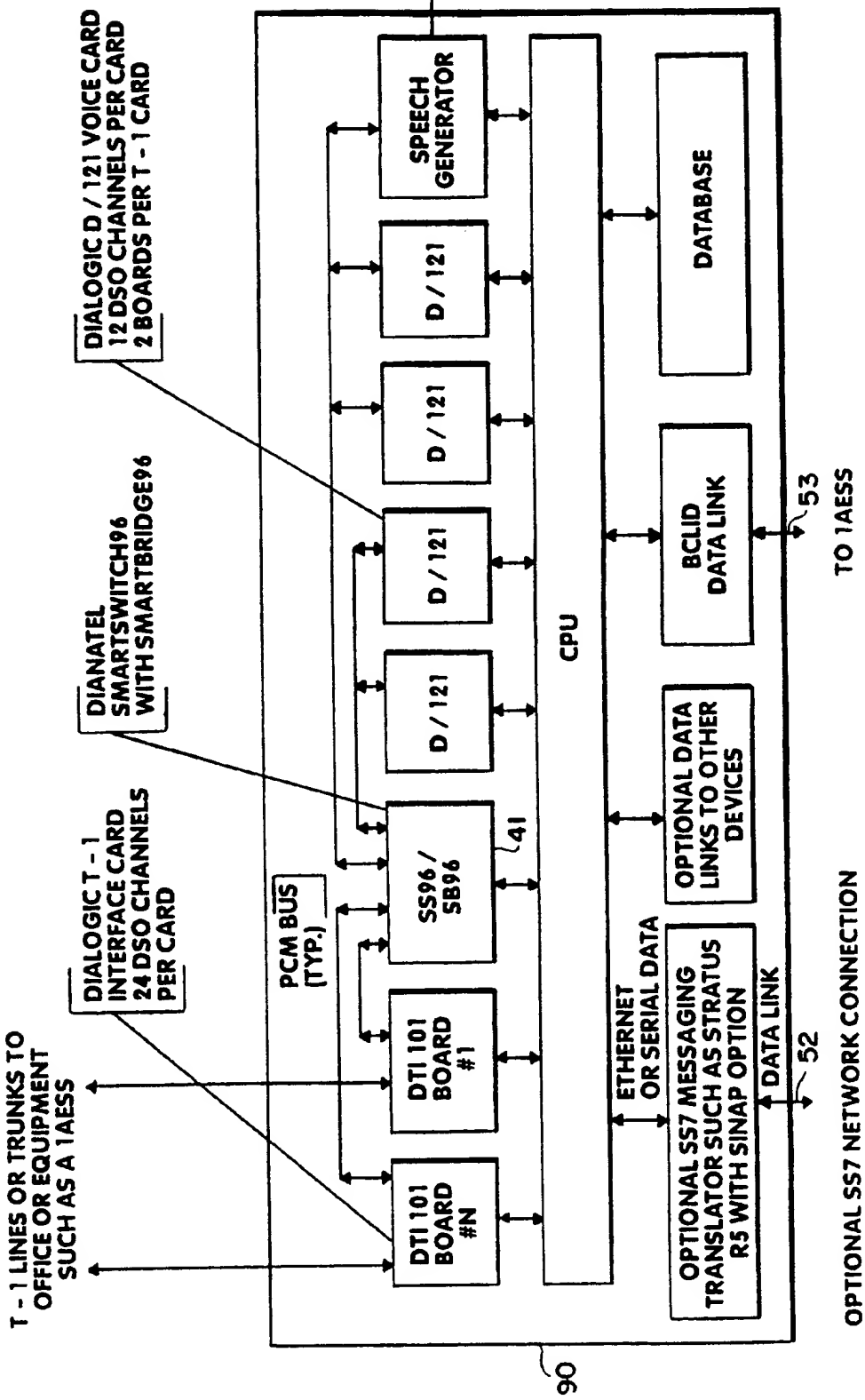

T - 1 LINES OR TRUNKS TO OFFICE OR EQUIPMENT SUCH AS A 1AESS
DIALOGIC T - 1 INTERFACE CARD 24 DSO CHANNELS PER CARD
DIANATEL SMARTSWITCH96 WITH SMARTBRIDGE96
DIALOGIC D / 121 VOICE CARD 12 DSO CHANNELS PER CARD 2 BOARDS PER T - 1 CARD
PCM BUS (TYP.)
DTI 101 BOARD #N
DTI 101 BOARD #1
SS96 / SB96
41
D / 121
D / 121
D / 121
D / 121
SPEECH GENERATOR
31K
712
CPU
DATABASE
BCLID DATA LINK
53
TO 1AESS
OPTIONAL DATA LINKS TO OTHER DEVICES
ETHERNET OR SERIAL DATA
OPTIONAL SS7 MESSAGING TRANSLATOR SUCH AS STRATUS R5 WITH SINAP OPTION
DATA LINK
52
OPTIONAL SS7 NETWORK CONNECTION
90

FIG. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US95/12318

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :H04Q 7/20
US CL :379/58
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 379/58, 59, 60, 61, 63, 57, 201, 202, 215, 217; 340/825.44; 455/33.1, 33.2

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X ---- Y	US, A, 5,090,051 (MUPPIDI ET AL) 18 February 1992, Figs 1-2 & Cols. 2-3.	1-13, 15, 17, 25-28, 38-39 ---------- 14, 16, 18-24, 29-37
Y	US, A, 5,307,399 (DAI ET AL) 26 April 1994, cols. 17-18.	14, 16, 18-22, 29-37
Y	US, A, 4,989,230 (GILLIG ET AL) 29 January 1991, Cols 6-7.	23-24
A	US, A, 5,175,758 (LEVANTO ET AL) 29 December 1992, see Abstract.	1-39
A	US, A, 4,748,655 (THROWER ET AL) 31 May 1988, see Abstract.	1-39

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 05 DECEMBER 1995	Date of mailing of the international search report 13 FEB 1996
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer MICHAEL B. CHERNOFF Telephone No. (703) 305-4750

Form PCT/ISA/210 (second sheet)(July 1992)*

This Page Blank (uspto)